EXHIBIT 1

Disclosure of Prior Art Combinations for Invalidity of U.S. Patent 5,961,601 ("'601 patent") under § 103

Claims of the '601 Patent	Prior Art Combinations
Claim 1	
[1.1] A computerized method for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client,	At least the following references disclose a method for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client: Danish (Ex. C-01); Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Graber (Ex. C-05); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Minor (Ex. C-10); Payne (Ex. C-11); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); WebStar (Ex. C-17); Unleashed (Ex. C-18); Admitted Prior Art (Ex. C-19); Yoshida (Ex. C-20); Yan (Ex. C-21); Chiu '022 (Ex. C-22); Lewine '565 (Ex. C-23); WWW-Talk (Ex. C-24); Benson (Ex. C-25); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
[1.2] said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation, the method comprising the steps of:	At least the following references disclose the services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation: Danish (Ex. C-01); Diener (Ex. C-02); DuFresne (Ex. C-03); Graber (Ex. C-05); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Minor (Ex. C-10); Payne (Ex. C-11); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); WebStar (Ex. C-17); Unleashed (Ex. C-18); Admitted Prior Art (Ex. C-19); Yoshida (Ex. C-20); Yan (Ex. C-21); Chiu '022 (Ex. C-22); Lewine '565 (Ex. C-23); WWW-Talk (Ex. C-24); Benson (Ex. C-25); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).

Claims of the '601 Patent	Prior Art Combinations
[1.3] the client initiating the conversation with the server using the stateless protocol;	At least the following references disclose the client initiating the conversation with the server using the stateless protocol: Danish (Ex. C-01); Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Graber (Ex. C-05); Ibrahim (Ex. C-06); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Minor (Ex. C-10); Payne (Ex. C-11); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); WebStar (Ex. C-17); Unleashed (Ex. C-18); Admitted Prior Art (Ex. C-19); Yoshida (Ex. C-20); Yan (Ex. C-21); Chiu '022 (Ex. C-22); Lewine '565 (Ex. C-23); WWW-Talk (Ex. C-24); Benson (Ex. C-25); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
[1.4] detecting when the request for a service requires preservation of the state information;	At least the following references disclose detecting when the request for a service requires preservation of the state information: Danish (Ex. C-01); Diener (Ex. C-02); DuFresne (Ex. C-03); Graber (Ex. C-05); Ibrahim (Ex. C-06); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Minor (Ex. C-10); Payne (Ex. C-11); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); WebStar (Ex. C-17); Unleashed (Ex. C-18); Admitted Prior Art (Ex. C-19); WWW-Talk (Ex. C-24); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
[1.5] performing said service and identifying all continuations in an output from said service, in response to said step of detecting;	At least the following references disclose performing said service and identifying all continuations in an output from said service, in response to said step of detecting: Danish (Ex. C-01); Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Graber (Ex. C-05); Ibrahim (Ex. C-06); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Minor (Ex. C-10); Payne (Ex. C-11); Popp (Ex. C-13); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); WebStar (Ex. C-17); Unleashed (Ex. C-18); Yoshida (Ex. C-20); Yan (Ex. C-21); Chiu '022 (Ex. C-22); Lewine '565 (Ex. C-23); WWW-Talk (Ex. C-24); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
[1.6] recursively embedding the state information in all identified continuations; and	At least the following references disclose recursively embedding the state information in all identified continuations: Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Ibrahim (Ex. C-06); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Minor (Ex. C-10); Payne (Ex. C-11); Popp (Ex. C-13); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); WebStar (Ex. C-17); Unleashed (Ex. C-18); Yoshida (Ex. C-20); Yan (Ex. C-21); Chiu '022 (Ex. C-22); Lewine '565 (Ex. C-23); WWW-Talk (Ex. C-24); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).

Claims of the '601 Patent	Prior Art Combinations
[1.7] communicating the output to the client, in response to said step of embedding, wherein the state information is preserved and provided to all services for the duration of the conversation.	At least the following references disclose communicating the output to the client, in response to said step of embedding, wherein the state information is preserved and provided to all services for the duration of the conversation: Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Ibrahim (Ex. C-06); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Payne (Ex. C-11); Popp (Ex. C-13); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); WebStar (Ex. C-17); Unleashed (Ex. C-18)); Yoshida (Ex. C-20); Yan (Ex. C-21); Chiu '022 (Ex. C-22); WWW-Talk (Ex. C-24); Benson (Ex. C-25); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
Claim 2	
[2.1] The method of claim 1, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding.	At least the following references disclose the method of claim 1, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding: Danish (Ex. C-01); Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Graber (Ex. C-05); Ibrahim (Ex. C-06); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Payne (Ex. C-11); Popp (Ex. C-13); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); WebStar (Ex. C-17); Unleashed (Ex. C-18); Yoshida (Ex. C-20); Yan (Ex. C-21); Chiu '022 (Ex. C-22); Lewine '565 (Ex. C-23); WWW-Talk (Ex. C-24); Benson (Ex. C-25); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
Claim 3	
[3.1] The method of claim 2, further comprising the step of storing at least part of the state information in a memory coupled to the server and	At least the following references disclose the method of claim 2, further comprising the step of storing at least part of the state information in a memory coupled to the server: Danish (Ex. C-01); Diener (Ex. C-02); Farquhar (Ex. C-04); Graber (Ex. C-05); Ibrahim (Ex. C-06); Levergood (Ex. C-07); Levine (Ex. C-08); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); Yoshida (Ex. C-20); Yan (Ex. C-21; Benson (Ex. C-25).
[3.2] wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations.	At least the following references disclose wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations: Diener (Ex. C-02); Farquhar (Ex. C-04); Lewine (Ex. C-09); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); Unleashed (Ex. C-18); Yan (Ex. C-21) WWW-Talk (Ex. C-24).
Claim 4	

Claims of the '601 Patent	Prior Art Combinations
[4.1] The method of claim 1, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client.	At least the following references disclose the method of claim 1, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client: DuFresne (Ex. C-03); Farquhar (Ex. C-04); Graber (Ex. C-05); Levine (Ex. C-08); Lewine (Ex. C-09); Popp (Ex. C-13); Tobin (Ex. C-15); Unleashed (Ex. C-18); and Williams (Ex. C-28) at 10:3-5 ("Unlike HTML, Java supports the notion of client-side validation, offloading appropriate processing onto the client for improved performance. Dynamic, real-time Web pages can be created. Using the above-mentioned custom UI components, dynamic Web pages can also be created."), 10:13-25 ("Sun's Java language has emerged as an industry-recognized language for 'programming the Internet.' Sun defines Java as: 'a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, dynamic, buzzword-compliant, general-purpose programming language. Java supports programming for the Internet in the form of platform-independent Java applets.' Java applets are small, specialized applications that comply with Sun's Java Application Programming Interface (API) allowing developers to add 'interactive content' to Web documents (e.g. simple animations, page adornments, basic games, etc.). Applets execute within a Java-compatible browser (e.g. Netscape Navigator) by copying code from the server to client. From a language standpoint, Java's core feature set is based on C++. Sun's Java literature states that Java is basically 'C++, with extensions from Objective C for more dynamic method resolution'."), and 12:55-13:42 ("FIG. 2 provides an alternative preferred embodiment in accordance with a Java based implementation of the invention. Java is a network application enabler for applications that utilize the Internet and HTML. Java is an object-oriented language that was developed by Sun Microsystems to speed development of their network applications. Only the differences between the native embodiment discussed earlier and the Java version is addressed to clarify the discussion. Accordingly, the browser 140, wallet manager 150, certificate manager 152, data manager 156, crypto & protocol library 157 and wallets 158 are the same as described in reference to FIG. 1B. A Payment Instruction Applet 200 at the Merchant Web Site 180 is responsible for delivering the order information to the PayWindow Applet 230 on the consumer's desktop 186. This order information has the same information that is contained in the Payment Instruction MIME message which was described in the PayWindow System View Native Code section. The Pay Instruction Applet 200 is a part of the payment HTML 192 page which is displayed by the merchant to the consumer. The Pay Instruction applet 200 requires that the following parameters be set with appropriate data when the payment page is generated by the merchant system GSO, Shipping Address, Merchant, Merchant Certificate,

Claims of the '601 Patent	Prior Art Combinations
	Merchant URL and Button Text. The Pay Instruction Applet 200 displays a button called 'Pay', 'Use PayWindow' or the value of the Button Text parameter. Once clicked, Pay Instruction Applet 200 delivers the above information to the Pay Window Applet 230 on the customer's desktop 186. The consumer interacts with the Pay window Applet 230 Applet to complete the payment transaction. The Address Requisition Applet 204 at the Merchant Web Site 180 is utilized by the merchant system 180 to obtain a consumer's shipping and/or billing address. This applet is displayed on the HTML page which accepts the consumer's shipping or billing address. The Address Requisition Applet 204 displays a button called 'V-wallet' or the value of the Button Text parameter. Once clicked, the address Requisition Applet 204 launches the AddressWindow applet 240 utilizing the consumer's address. The AddressWindow applet 240 interacts with the consumer and send the address information to the merchant system 180. The address information is then processed consistent with the processing in the native system. The Certificate Issuance CGI scripts 162 and the Single Account Wallet 160 at the Bank Web Site 182 is processed as described in the native system. The Certificate Installation Applet 220 of the Bank Web Site 182 is utilized by the Certificate Issuance CGI scripts 162 system to deliver a consumer's certificate to the consumer's desktop. A variety of applets are provided at the Veri Fone web site 184 to make the consumer's shopping experience easy and efficient. These helper applications include a Setup Applet 212, PayWindow Applet 210 and an AddressWindow Applet 214 similar to the PayWindow Helper Application Native section discussed above.").
Claim 5	
[5.1] The method of claim 4, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information.	At least the following references disclose the method of claim 4, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information: Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Perrochon (Ex. C-12); Popp (Ex. C-13); Tobin (Ex. C-15); Unleashed (Ex. C-18).
Claim 6	
[6.1] The method of claim 1,	At least the following references disclose the method of claim 1, further comprising the steps of: the

Claims of the '601 Patent	Prior Art Combinations
further comprising the steps of: the client selecting a second continuation from said all identified continuations with embedded state information; and	client selecting a second continuation from said all identified continuations with embedded state information: Danish (Ex. C-01); Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Graber (Ex. C-05); Levergood (Ex. C-07); Levine (Ex. C-08); Popp (Ex. C-13); Da Silva (Ex. C-14); WebStar (Ex. C-17); Unleashed (Ex. C-18); Lewine '565 (Ex. C-23); WWW-Talk (Ex. C-24); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
[6.2] restoring the state information from said second continuation and invoking an associated second service with restored state information;	At least the following references disclose restoring the state information from said second continuation and invoking an associated second service with restored state information: Danish (Ex. C-01); Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Graber (Ex. C-05); Levergood (Ex. C-07); Levine (Ex. C-08); Popp (Ex. C-13); Da Silva (Ex. C-14); Unleashed (Ex. C-18); Yan (Ex. C-21); WWW-Talk (Ex. C-24); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
[6.3] recursively identifying and embedding the state information in all continuations associated with an output from said second service.	At least the following references disclose recursively identifying and embedding the state information in all continuations associated with an output from said second service: Danish (Ex. C-01); Diener (Ex. C-02); DuFresne (Ex. C-03); Farquhar (Ex. C-04); Graber (Ex. C-05); Levine (Ex. C-08); Popp (Ex. C-13); Da Silva (Ex. C-14); Unleashed (Ex. C-18); Yan (Ex. C-21); WWW-Talk (Ex. C-24); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
Claim 7	
[7.1] The method of claim 1, further comprising the step of correlating the state information to a specific conversation.	At least the following references disclose the method of claim 1, further comprising the step of correlating the state information to a specific conversation: Diener (Ex. C-02); Farquhar (Ex. C-04); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Minor (Ex. C-10); Payne (Ex. C-11); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); OCLC Gateway (Ex. C-16); Unleashed (Ex. C-18); Yan (Ex. C-21); WWW-Talk (Ex. C-24); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
Claim 8	
[8.1] The method of claim 1, wherein the client and the	At least the following references disclose wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are

Claims of the '601 Patent	Prior Art Combinations
server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	hyperlinks to one of hypertext markup language files and common gateway interface programs: Danish (Ex. C-01); Diener (Ex. C-02); Farquhar (Ex. C-04); Graber (Ex. C-05); Ibrahim (Ex. C-06); Levergood (Ex. C-07); Levine (Ex. C-08); Lewine (Ex. C-09); Minor (Ex. C-10); Payne (Ex. C-11); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); Tobin (Ex. C-15); OCLC Gateway (Ex. C-16); WebStar (Ex. C-17); Unleashed (Ex. C-18); Yoshida (Ex. C-20); Yan (Ex. C-21); Chiu '022 (Ex. C-22); WWW-Talk (Ex. C-24); Benson (Ex. C-25); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
Claim 9	
[9.1] The method of claim 8, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	At least the following references disclose the method of claim 8, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria: Danish (Ex. C-01); Levergood (Ex. C-07); Lewine (Ex. C-09); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); OCLC Gateway (Ex. C-16); Unleashed (Ex. C-18); Amazon 1995 Website (Ex. C-27)
Claim 10	
[10.1] The method of claim 8, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	At least the following references disclose the method of claim 8, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria: Danish (Ex. C-01); Graber (Ex. C-05); Levergood (Ex. C-07); Lewine (Ex. C-09); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); OCLC Gateway (Ex. C-16); Unleashed (Ex. C-18); Yan (Ex. C-21); WWW-Talk (Ex. C-24); Amazon 1995 Website (Ex. C-27).
Claim 11	
[11.1] The method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is a request for an HTML file to	At least the following references disclose the method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments: Diener (Ex. C-02); DuFresne (Ex. C-03); Graber (Ex. C-05); Ibrahim (Ex. C-06); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); WebStar (Ex. C-17); Unleashed (Ex. C-18); Admitted Prior Art (Ex. C-19); Yoshida (Ex. C-20); Chiu '022 (Ex. C-22); WWW-Talk (Ex. C-24); Benson (Ex. C-25); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).

Claims of the '601 Patent	Prior Art Combinations
invoke a CGI converter program with the identified continuation and the state information passed as arguments.	
Claim 12	
[12.1] The method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program.	At least the following references disclose the method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program: Diener (Ex. C-02); DuFresne (Ex. C-03); Graber (Ex. C-05); Ibrahim (Ex. C-06); Perrochon (Ex. C-12); Popp (Ex. C-13); Da Silva (Ex. C-14); WebStar (Ex. C-17); Unleashed (Ex. C-18); Admitted Prior Art (Ex. C-19); Yoshida (Ex. C-20); WWW-Talk (Ex. C-24); Benson (Ex. C-25); Spinning the Web (Ex. C-26); Amazon 1995 Website (Ex. C-27).
Claim 14	
[14.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to provide	*See* claim 1.1.
[14.2] a method for preserving state information in a conversation between a client adapted to request services from one or more	*See* claim 1.1.

Claims of the '601 Patent	Prior Art Combinations
servers which are networked via a stateless protocol to the client,	
[14.3] said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation, the method comprising the steps of:	*See* claim 1.2.
[14.4] the client initiating the conversation with the server using the stateless protocol;	*See* claim 1.3
[14.5] detecting when the request for a service requires preservation of the state information;	*See* claim 1.4
[14.6] performing said service and identifying all	*See* claim 1.5

Claims of the '601 Patent	Prior Art Combinations
continuations in an output from said service, in response to said step of detecting;	
[14.7] recursively embedding the state information in all identified continuation; and	*See* claim 1.6
[14.8] communicating the output to the client, in response to said step of embedding; wherein the state information is preserved and provided to all services for the duration of the conversation.	*See* claim 1.7
Claim 15	
[15.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 14, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding.	*See* claim 2.1.
Claim 16	
[16.1] A program storage	*See* claim 3.1

Claims of the '601 Patent	Prior Art Combinations
device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 15, further comprising the step of storing at least part of the state information in a memory coupled to the server and	
[16.2] wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations.	*See* claim 3.2.
Claim 17	
[17.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 14, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of	*See* claim 4.1

Claims of the '601 Patent	Prior Art Combinations
communicating the output to the client.	
Claim 18	
[18.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 14, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information.	*See* claim 5.1.
Claim 19	
[19.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 14, further comprising the steps of: the client selecting a second continuation from said all identified continuations with	*See* claims 6.1-6.3.

Claims of the '601 Patent	Prior Art Combinations
embedded state information; and restoring the state information from said second continuation and invoking an associated second service with restored state information; recursively identifying and embedding the state information in all continuations associated with an output from said second service.	
Claim 20	
[20.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 14, further comprising the step of correlating the state information to a specific conversation.	*See* claim 7.1.
Claim 21	
[21.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by	*See* claim 8.1.

Claims of the '601 Patent	Prior Art Combinations
the computer to perform method steps as claimed in claim 14, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	
Claim 22	
[22.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 21, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	*See* claim 9.1.
Claim 23	
[23.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by	*See* claim 10.1.

Claims of the '601 Patent	Prior Art Combinations
the computer to perform method steps as claimed in claim 21, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	
Claim 24	
[24.1] The program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 21, wherein said step of embedding further comprises the step of: modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments.	*See* claim 11.1.
Claim 25	
[25.1] The program storage device readable by a computer, tangibly embodying a program of	*See* claim 12.1

Claims of the '601 Patent	Prior Art Combinations
instructions executable by the computer to perform method steps as claimed in claim 21, wherein said step of embedding further comprises the step of: modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program.	
Claim 27	
[27.1] A computer system for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client,	*See* claim 1.1.
[27.2] said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of	*See* claim 1.2.

Claims of the '601 Patent	Prior Art Combinations
communications between the client and one or more servers for said services, wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation, the system comprising:	
[27.3] the client being adapted for initiating a conversation with the server using the stateless protocol;	*See* claim 1.3.
[27.4] state detection logic for detecting when the request for a service requires preservation of the state information;	*See* claim 1.4.
[27.5] search logic for identifying all continuations in an output from said service, in response to said step of detecting;	*See* claim 1.5.
[27.6] converter logic for recursively embedding the state information in all identified continuations; and	*See* claim 1.6.
[27.7] communication logic for communicating the	*See* claim 1.7.

Claims of the '601 Patent	Prior Art Combinations
output to the client; wherein the state information is preserved and provided to all services for the duration of the conversation.	
Claim 28	
[28.1] The computer system of claim 27, wherein said converter logic is executed by the server and said communication logic communicates the output with embedded state information from the server to the client.	*See* claim 2.1.
Claim 29	
[29.1] The computer system of claim 28, further comprising: a memory, coupled to the server, for storing at least part of the state information;	*See* claim 3.1.
[29.2] wherein said converter logic is adapted for embedding an index representing said part of the state information in said all identified continuations.	*See* claim 3.2.
Claim 30	
[30.1] The computer system of claim 27, wherein said communication logic	*See* claim 4.1.

Claims of the '601 Patent	Prior Art Combinations
communicates the output without embedded state information from the server to the client; and wherein the server is adapted for dynamically downloading said converter logic to the client for execution.	
Claim 31	
[31.1] The computer system of claim 30, further comprising: a memory, coupled to the client, for storing at least part of the state information; wherein said converter logic is further adapted for embedding an index representing said part of the state information.	*See* claim 5.1.
Claim 32	
[32.1] The computer system of claim 27, wherein the client selects a second continuation from said all identified continuations with embedded state information, further comprising: the converter logic being further adapted for restoring the state information from said second continuation,	*See* claims 6.1-6.3.

Claims of the '601 Patent	Prior Art Combinations
invoking an associated second service with restored state information, and recursively identifying and embedding the state information in all continuations associated with an output from said second service.	
Claim 33	
[33.1] The computer system of claim 27, wherein the state information is correlated to a specific conversation.	*See* claim 7.1.
Claim 34	
[34.1] The computer system of claim 27, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	*See* claim 8.1.
Claim 35	
[35.1] The computer system of claim 34, further comprising filter logic for filtering one of said	*See* claim 9.1.

Claims of the '601 Patent	Prior Art Combinations
hyperlinks and data output from said services according to a predetermined criteria.	
Claim 36	
[36.1] The computer system of claim 34, further comprising integration logic for adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	*See* claim 10.1.
Claim 37	
[37.1] The computer system of claim 34, wherein said step of embedding further comprises the step of: modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments.	*See* claim 11.1.
Claim 38	
[38.1] The computer system of claim 34, wherein said step of embedding further comprises the step of: modifying an identified continuation which is an	*See* claim 12.1.

Claims of the '601 Patent	Prior Art Combinations
invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program.	
Claim 40	
[40.1] A computer system for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client,	*See* claim 1.1.
[40.2] said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein	*See* claim 1.2.

Claims of the '601 Patent	Prior Art Combinations
the client must invoke one of the continuations to continue the conversation, the system comprising:	
[40.3] the client being adapted for initiating the conversation with the server using the stateless protocol;	*See* claim 1.3.
[40.4] state detection means for detecting when the request for a service requires preservation of the state information;	*See* claim 1.4.
[40.5] search means for identifying all continuations in an output from said service, in response to said step of detecting;	*See* claim 1.5.
[40.6] converter means for recursively embedding the state information in all identified continuations; and	*See* claim 1.6.
[40.7] communication means for communicating the output to the client; wherein the state information is preserved and provided to all services for the duration of the conversation.	*See* claim 1.7.
Claim 41	
[41.1] The computer system	*See* claim 2.1.

Claims of the '601 Patent	Prior Art Combinations
of claim 40, wherein said converter means is executed by the server and said communication means communicates the output with embedded state information from the server to the client.	
Claim 42	
[42.1] The computer system of claim 41, further comprising: a memory, coupled to the server, for storing at least part of the state information;	*See* claim 3.1.
[42.2] wherein said converter means is adapted for embedding an index representing said part of the state information in said all identified continuations.	*See* claim 3.2.
Claim 43	
[43.1] The computer system of claim 40, wherein said communication means communicates the output without embedded state information from the server to the client; and wherein the server is adapted for dynamically downloading said converter means to the	*See* claim 4.1.

Claims of the '601 Patent	Prior Art Combinations
client for execution.	
Claim 44	
[44.1] The computer system of claim 43, further comprising: a memory, coupled to the client, for storing at least part of the state information; wherein said converter means is further adapted for embedding an index representing said part of the state information.	*See* claim 5.1.
Claim 45	
[45.1] The computer system of claim 41, wherein the client selects a second continuation from said all identified continuations with embedded state information, further comprising: the converter means being further adapted for restoring the state information from said second continuation, invoking an associated second service with restored state information, and recursively identifying and embedding the state information in all continuations associated	*See* claims 6.1-6.3.

Claims of the '601 Patent	Prior Art Combinations
with an output from said second service.	
Claim 46	
[46.1] The computer system of claim 45, further comprising integration means for adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	*See* claim 10.1.
Claim 47	
[47.1] The computer system of claim 40, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	*See* claim 8.1.
Claim 48	
[48.1] The system of claim 47, wherein said converter means further comprises: means for modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the	*See* claim 11.1.

Claims of the '601 Patent	Prior Art Combinations
identified continuation and the state information passed as arguments.	
Claim 49	
[49.1] The system of claim 47, wherein said converter means further comprises: means for modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said converter means comprises the CGI program.	*See* claim 12.1.
Claim 51	
[51.1] A computerized method for preserving state information in a conversation via a stateless protocol between a client adapted to request services from one or more servers, the method comprising the steps of:	*See* claim 1.1.
[51.2] receiving a service request including state information, via the stateless protocol;	*See* claim 1.3.
[51.3] identifying all continuations in an output	*See* claims 1.5-1.6.

Claims of the '601 Patent	Prior Art Combinations
from said service and recursively embedding the state information in all identified continuations, in response to said request; and	
[51.4] communicating a response including the continuations and embedded state information, wherein the continuations enable another service request and one of the continuations must be invoked to continue the conversation.	*See* claim 1.7.
Claim 52	
[52.1] The method of claim 51, wherein said embedding is performed by a server and said step of communicating is in response to said embedding step.	*See* claim 2.1.
Claim 53	
[53.1] The method of claim 52, further comprising the step of storing at least part of the state information in a memory coupled to the server and	*See* claim 3.1.
[53.2] wherein embedding step includes embedding an index representing said part of the state information in	*See* claim 3.2.

Claims of the '601 Patent	Prior Art Combinations
said all identified continuations.	
Claim 54	
[54.1] The method of claim 51, further comprising the step of dynamically downloading computer program code to the client to perform said embedding step, in response to said step of communicating the output to the client.	*See* claim 4.1.
Claim 55	
[55.1] The method of claim 54, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said embedding step includes embedding an index representing said part of the state information.	*See* claim 5.1.
Claim 56	
[56.1] The method of claim 51, further comprising the steps of: receiving a second request associated with a second continuation from said all identified continuations with embedded state information;	*See* claims 6.1-6.3.

Claims of the '601 Patent	Prior Art Combinations
and restoring the state information from said second continuation and invoking an associated second service with restored state information; recursively identifying and embedding the state information in all continuations associated with an output from said second service.	
Claim 57	
[57.1] The method of claim 51, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	*See* claim 8.1.
Claim 58	
[58.1] The method of claim 57, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	*See* claim 9.1.

Claims of the '601 Patent	Prior Art Combinations
Claim 59	
[59.1] The method of claim 57, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	*See* claim 10.1.
Claim 60	
[60.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to provide	*See* claim 14.1.
[60.2] a method for preserving state information in a conversation via a stateless protocol between a client adapted to request services from one or more servers, the method comprising the steps of:	*See* claim 1.1.
[60.3] receiving a service request including state information, via the stateless protocol;	*See* claim 1.3.
[60.4] identifying all continuations in an output from said service and recursively embedding the state information in all	*See* claims 1.5-1.6.

Claims of the '601 Patent	Prior Art Combinations
identified continuations, in response to said request; and	
[60.5] communicating a response including the continuations and embedded state information, wherein the continuations enable another service request and one of the continuations must be invoked to continue the conversation.	*See* claim 1.7.
Claim 61	
[61.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 60, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding.	*See* claim 2.1.
Claim 62	
[62.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform	*See* claim 3.1.

Claims of the '601 Patent	Prior Art Combinations
method steps as claimed in claim 60, further comprising the step of storing at least part of the state information in a memory coupled to the server and	
wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations.	*See* claim 3.2.
Claim 63	
[63.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 60, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client.	*See* claim 4.1.
Claim 64	
[64.1] A program storage device readable by a	*See* claim 5.1.

Claims of the '601 Patent	Prior Art Combinations
computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 60, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information.	
Claim 65	
[65.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 60, further comprising the steps of: receiving a second request associated with a second continuation from said all identified continuations with embedded state information; and restoring the state information from said	*See* claims 6.1-6.3.

Claims of the '601 Patent	Prior Art Combinations
second continuation and invoking an associated second service with restored state information; recursively identifying and embedding the state information in all continuations associated with an output from said second service.	
Claim 66	
[66.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 60, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	*See* claim 8.1.
Claim 67	
[67.1] A program storage device readable by a computer, tangibly	*See* claim 9.1.

Claims of the '601 Patent	Prior Art Combinations
embodying a program of instructions executable by the computer to perform method steps as claimed in claim 66, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	
Claim 68	
[68.1] A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps as claimed in claim 66, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	*See* claim 10.1.

EXHIBIT 2

SPINNING THE WEB

A GUIDE TO SERVING INFORMATION ON THE WORLD WIDE WEB

YUVAL FISHER

```
    echo ""
    cat  $frame
    echo ""
  end
  echo "--BoundaryString--"
```

9.10.5 Maintaining State – a Shopping Cart

Many vendors on the WWW display their wares, sometimes even with care and taste. Many, however, require the purchaser, already unhappy about parting with his or her money, to remember an item number and price and enter it on a separate page. What these electronic merchants need is a way to add items to a "shopping cart" that the purchaser can view and alter. This is not completely simple, because of the stateless server/client model of the WWW. That is, the server must remember who the client is, even though it is just serving documents to Internet hosts, without knowing the user on the other side.

The easy way to do this is to force the user to authenticate himself, using the authentication schemes described in Chapter 4. The user is then identified by the `REMOTE_USER` environment variable. But authentication is an extra (ugly) step between the merchant and the purchaser's money, so we present a different solution.[12] Incidentally, schemes such as this are useful for more than just shopping baskets. See, for example, the game of adventure at [95].[13]

The state mechanism we describe here assigns a special number to each client. These are assigned when the client first connects. This number is included in all the URLs that the client sees after the initial connection, and hence all document requests to the server contain this identifying number

[12]It is also possible to use the `Set-Cookie:` HTTP header as described in Chapter 3, but not all browsers will return a `Cookie` header, so this is not a universal solution.

[13]`http://inls.ucsd.edu/y/OhBoy/Adventure/`

in the submitted URL. This is good. It means that the server (that is, the CGI scripts used) can recognize to whom they are sending data by looking at their URL and extracting the unique number assigned to the specific client. Unfortunately, all the HTML served must be custom generated on the fly for each client. This is bad. Some more good and bad news is that commercial server products largely automate this whole process, but they cost a lot more than the widely available free servers.

To make life simple (both in this example and in the real world), we will create HTML templates that will be filtered for the addition of the client's identifier and then served. We use the special string ### as a token that is replaced by the client's identifier before the template is served. This scheme has the added advantage that the store pages can be viewed in any browser, for easy checking of modifications (which is not the case if the HTML is included in the code of the script).

We will require two CGI scripts. The first, `see.pl`, serves a document that includes a client's identifier. It finds the template to be served in `PATH_INFO`, and it finds the client's identifier in `QUERY_STRING`. It then filters the template, adding the identifier, and serves it up.

The second script, `cart.pl`, accepts items to be added to the cart and displays its contents. The cart itself is just a file that is stored on the server and which contains a list of the items added to it. In this example, we do not bother with prices, taxes, shipping, and quantities.

Note: This scheme really only makes sense for stores (or whatever) with more than one page of items, since if there is only one page, it is possible to have a single form with checkboxes (that is `<INPUT TYPE="checkbox">` for the desired items).

The Store Entrance

The topmost page is an entrance, `enter.html` from which we call the CGI script `see.pl` with a special value, `setup`, that will

be put in QUERY_STRING. When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier. The URL referring to see.pl also contains the name of a file, in this case store1.html, in which see.pl replaces occurrences of ### with the client identifier. This file is then served.

Enter.html looks like:

```
<HTML> <!-- enter.html: the store front -->
<HEAD>
 <TITLE>Welcome to the Nothing Store</TITLE>
</HEAD>
<BODY>
 <H2>Welcome to the Nothing Store</H2>
 <A HREF="/cgi-bin/see.pl/store1.html?setup">
      Enter our Store</A>.
 <HR>
 <EM>Nothing Productions</EM>
</BODY>
</HTML>
```

If the client navigates back to this entrance and selects the "Enter our Store" link again, it will be assigned a new client identifier and hence a new shopping cart. This can be avoided, to some extent, by insisting that the store entrance have the URL /cgi-bin/see.pl/store1.html?setup (either by telling people that this is the entrance, or by repeated clicking of the heels).

The Stores

We call the template store1.html, even though we might prefer to give it some different name, such as store1.template. A different name would have the advantage of distinguishing templates from the final HTML that is served (even though these templates contain perfectly respectable HTML). The reason we avoided a different name is that unless we adjust the local .mime.types file (see Section 2.3), browsers will not

recognize the template files as HTML, and this would make it more painful to view the files.

The template `store1.html`, rendered in Figure 9.4, looks like:

```
<HTML> <!-- store1.html: The first store window -->
<HEAD>
 <TITLE>Welcome to Store One</TITLE>
</HEAD>
<BODY>
 <H2>This is Store One</H2>
 <H3> The first item is Nothing </H3>
 <A HREF=/cgi-bin/cart.pl/store1_item1?###>Add to Cart</a>.
 <H3> The second item is Nothing </H3>
 <A HREF=/cgi-bin/cart.pl/store1_item2?###>Add to Cart</a>.
 <P>
 <H4><A HREF=/cgi-bin/cart.pl?###>See Cart Contents</a></H4>
 <P>
 See our
 <A HREF=/cgi-bin/see.pl/store2.html?###>other items</A>.
 <HR>
 <EM>Nothing Productions</EM>
</BODY>
</HTML>
```

In `store1.html` there are two types of hyperlinks. One, near the bottom, calls `see.pl` in such a way that `QUERY_STRING` will be set to the client identifier (recall that the `###` was replaced with it when `see.pl` was called the first time with `QUERY_STRING=setup`). In that call, `PATH_INFO=/store2.html`, which means that following this link will cause `see.pl` to filter the contents of `store2.html`, again replacing the token `###` with the client identifier. We do not show `store2.html` here, but it can be essentially identical to `store1.html`, with ones and twos exchanged.

The second type of hyperlink in `store.html` calls the CGI script `cart.pl`. This script also receives the client identifier in `QUERY_STRING`, and it receives a string, e.g., `store1_item1` in

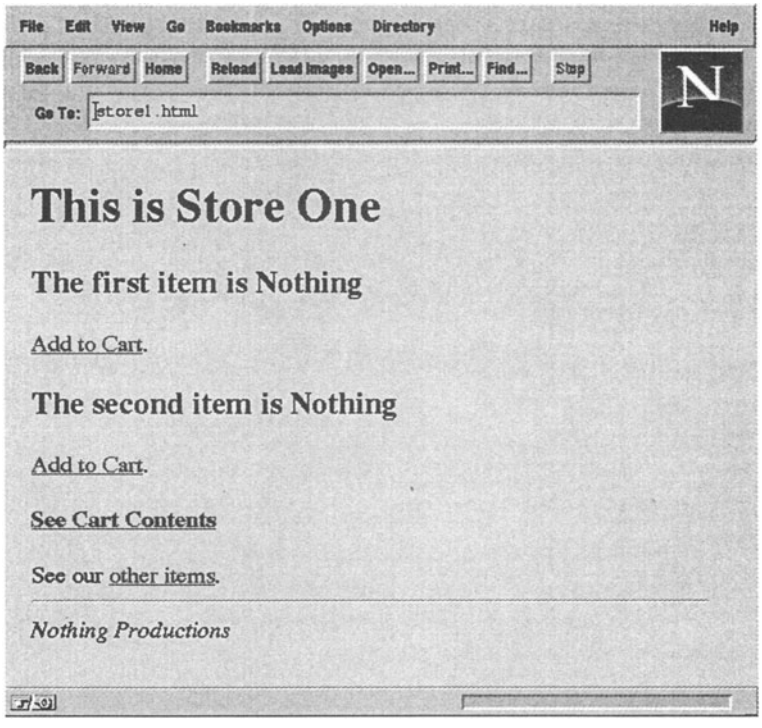

FIGURE 9.4 A store containing (somewhat unenticing) items that can be added to a shopping cart.

PATH_INFO, that tells it which item was put in the shopping cart.

Either way, the client identifier is included in all URLs in the version of `store.html` that is passed through `see.pl`. Moreover, all these URLs must reference CGI scripts (not static HTML pages), so that the client identifier can be propagated through whatever other pages are displayed.

Finally, the "add to cart" hyperlink can be a form. This would have the advantage of allowing extra information to be included in the request (for example, in `<INPUT TYPE="hidden">` tags).

The Scripts

Here is the `see.pl` Perl script:

```
#!/usr/local/bin/perl
```

```
# see.pl                                                    # 2
# A Perl program to extract a file name from the            # 3
# PATH_INFO, fetch the file, filter it (adding the          # 4
# client identifier), and serve it up.                      # 5
#                                                           # 6
# The next two executable lines are site-dependent.         # 7
# The actual path to the files.                             # 8
$realpath="/usr/people/fisher/HTML/Test/";                  # 9
# A list of files that can be served by this script.        # 10
@ok_to_serve=('store1.html','store2.html');                 # 11
#                                                           # 12
# Print out header.                                         # 13
print "Content-Type: text/html\n\n";                        # 14
#                                                           # 15
# Find the file name.                                       # 16
$filename = substr($ENV{'PATH_INFO'},1);                    # 17
# Find the client identifier name                           # 18
$clientid = $ENV{'QUERY_STRING'};                           # 19
#                                                           # 20
# Ensure we are allowed to serve the file.                  # 21
if (!grep(/^$filename$/,@ok_to_serve)) {                    # 22
    print "Bad file name\n";    # Complaint could be       # 23
    exit(1);                    # more informative.        # 24
}                                                           # 25
#                                                           # 26
# When entering the store, create a new client              # 27
# identifier. We use the time + process id.                 # 28
if ($clientid eq "setup") {$clientid = time . $$;}          # 29
#                                                           # 30
# Check for funny characters in clientid.                   # 31
if ($clientid =~ /\D/) {                                    # 32
    print "Have we met ?\n";    # Complaint could be       # 33
    exit(1);                    # more informative.        # 34
}                                                           # 35
#                                                           # 36
# Prepend the real path.                                    # 37
$filename = $realpath.$filename;                            # 38
```

```
#                                                          # 39
if (!open(HTMLFILE,$filename)) {                           # 40
            print "Can't open template, sorry.\n";         # 41
            exit(1);                                       # 42
}                                                          # 43
$/="</HTML>";                    # read the whole          # 44
$everything = <HTMLFILE>;        # file at once.           # 45
close(HTMLFILE);                                           # 46
#                                                          # 47
# Substitute the client identifier in the template.        # 48
$everything =~ s/###/$clientid/g;                          # 49
print $everything;                                         # 50
```

Important: Note that this script opens local files that are passed in `PATH_INFO` (copied into `$filename`). This is a potential security hole, and it is very important to make sure that the string containing the file name is kosher. In this example, we choose the most paranoid and secure of methods: allowing only a known list of files (specified in `@ok_to_serve`) to be served. We could change lines 21–26 to be:

```
# Check for bad (possibly insecure) characters
if ($filename =~ m+[';><\/\|>\*]+) { # use m+ to
  print "Bad file name\n";          # match slashes
  exit(1);
}
```

in which case the whole content of `$realpath` could be served relatively securely. In this case, we check that `$filename` does not contain potentially evil characters. For example, the "|" character denotes the opening of a pipe, and that allows essentially any command to be executed on the server.

Some things to note about this code are:

- When `QUERY_STRING=setup`, we create a new client identifier (line 29) using the current time (measured in seconds since January 1, 1970) prepended to the process id, ensuring that simultaneous accesses will have different identifiers. This is a bit transparent and can lead to

abuse by people who guess client identifiers in order to view other client's carts. A different approach could be to encrypt the user's Internet host name (along with other unique information to avoid repeated identifiers). This would have the extra benefit of restricting potential abuse to people (who appear to be) from the same host.

- Line 32 checks that the client identifier is numerical – which it would not be with the modification suggested above. Lines 23 and 33 are executed only in the case of tampering with the URLs (for the paranoid) or transmission error (for the optimistic). Those are good places to place warnings in a log that the store-meister can peruse when searching for potential security abuse.

- Notice that when errors are printed, the resulting HTML does not contain any of the <HTML>, <HEAD>, or <BODY> elements. It would be cleaner to include these.

The shopping cart consists of a file whose name is just the client identifier. Thus, each client gets a unique identifier and a unique cart. Unlike the stores, the cart is not a template. The complete HTML for the cart is generated when the request is served. It is possible, though, to create a template and simply replace a token with the cart contents just as we did with the `store1.html` template.

The `cart.pl` script looks like:

```
#!/usr/local/bin/perl
# cart.pl                                                    # 2
# A Perl program to add an item to a shopping cart.          # 3
#                                                            # 4
# The next three executable lines are site-dependent.        # 5
# This is the path to the directory in which the cart        # 6
# data is stored.                                            # 7
$cartpath="/usr/people/fisher/HTML/Test/";                   # 8
#                                                            # 9
# This is a URL to a page that takes personal                # 10
# information (credit # card number, whatever) and           # 11
```

```
# finalizes the transaction.                                    # 12
$payURL="http://host/path";                                     # 13
#                                                               # 14
# This is a store URL for returning to shopping.                # 15
$storeURL="/cgi-bin/see.pl/store1.html";                        # 16
#                                                               # 17
#                                                               # 18
# Print out header                                              # 19
print "Content-Type: text/html\n\n";                            # 20
#                                                               # 21
# Find the item added                                           # 22
$item = substr($ENV{'PATH_INFO'},1);                            # 23
# Find the client identifier name                               # 24
$clientid = $ENV{'QUERY_STRING'};                               # 25
#                                                               # 26
# Check for bad characters and complain                         # 27
if ($item=~ m/['';<>\|\*]/) {                                   # 28
    print "Bad item\n";              # This could be           # 29
    exit();                          # more informative        # 30
}                                                               # 31
#                                                               # 32
# Check for funny characters                                    # 33
if ($clientid =~ m/\D/) {                                       # 34
    print "Have we met ?\n";         # This could be           # 35
    exit(1);                         # more informative        # 36
}                                                               # 37
#                                                               # 38
# Prepend the real path                                         # 39
$filename = $cartpath.$clientid;                                # 40
if ($item) {                         # append item to          # 41
    open(CARTFILE,">>".$filename);# cart if there              # 42
    print CARTFILE "$item\n";        # is one.                 # 43
    close CARTFILE;                                             # 44
}                                                               # 45
#                                                               # 46
# Now display cart content                                      # 47
if (!open(CARTFILE,$filename)) {                                # 48
```

```
    print "Can't open cart, sorry... \n";              # 49
    exit(1);                    # should be more     # 50
}                               #  helpful.          # 51
#                                                     # 52
# This is the cart HTML.                              # 53
print "<HTML>";                                       # 54
print "\n<HEAD><TITLE>Shopping Cart</TITLE></HEAD>";  # 55
print "\n<BODY>\n<H3>Cart Contents:</H3>\n<UL>";      # 56
while(<CARTFILE>) {                                   # 57
    s/_/ /g;                    # change _ to space   # 58
    print "\n<LI> $_";                                # 59
}                                                     # 60
close CARTFILE;                                       # 61
print "\n</UL>";                                      # 62
print "\n<A HREF=\"$storeURL?$clientid\">";           # 63
print "Return To Shopping</A>";                       # 64
print "\n<P>\n<A HREF=\"$payURL?$clientid\">";        # 65
print "Pay for items</A>";                            # 66
print "\n</BODY><\HTML>"                              # 67
```

Some things to note about this code are:

- This code must deal with the same security issue as `see.pl`. In this case, we open a file whose name is the client identifier. Since we selected numerical client identifiers, it is easy to avoid insecure characters in the file name.

- It is impossible to prevent a user from guessing cart names. Worse, a malicious user could fill up the `$cart¬path` directory with lots of files containing gigantic items. A semi-fix to this, or at least a bottleneck for the malicious user, would be to validate `$cartname` with a list of client identifiers created by `see.pl`. `See.pl` could then balk at creating too many client identifiers in a short time for the same Internet host. This is a lot of work for improbable abuse.

- The check on line 28 is not strictly necessary. On the other hand, it could be strengthened to check that the item belongs to a list of valid items.
- In a large store, it would be polite to allow the shopper to return to the section of the store from which he or she added the item. Currently, the `$storeURL` (lines 16 and 63) is static, allowing the client to return to a fixed URL. It would be nicer to include a reference to the store (in the `PATH_INFO`), or in a hidden input item if the "add to cart" link was actually a form. This reference could then be used for `$storeURL`.
- The `$payURL` (line 65) should refer to a script that adds up the cost of the items, allows the user to enter a credit card number (or whatever), and completes the transaction. Note that the client identifier is (and must be) passed to this script.
- When the item is non-empty, on line 41, we add it to the cart. An empty item allows us to display the cart without modifying it. This is what happens in the "See Cart Contents" hyperlink in `store1.html`.
- Note that this code does not allow users to take things out of the cart (too bad it can't force them to pay). This can be done with a separate script linked to a "remove from cart" button, or, better yet, a script that updates the cart and allows the user to modify the quantity of each item that appears in the cart.

The result of calling `cart.pl` a few times looks like:

```
<HTML>
<HEAD><TITLE>Shopping Cart</TITLE></HEAD>
<BODY>
<H3>Cart Contents:</H3>
<UL>
<LI> store1 item1
```

```
<LI> store1 item2
</UL>
<A HREF="/cgi-bin/see.pl/store1.html?8523765492046">\
Return To Shopping</A>
<P>
<A HREF="http://host/path?8523765492046">Pay for items</A>
</BODY><HTML>
```

Further Reading

The WWW security FAQ (see [235][14]) contains another discussion about CGI security, as well as other WWW security issues. A variety of CGI programming tools in various languages are discussed in Section 11.10.

A tutorial on Perl programming with CGI can be found at [90].[15] Another tutorial with many nice examples in Perl, Pascal, C, and C++ can be found at [241].[16]

[14] http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html
[15] http://www.catt.ncsu.edu/~bex/tutor/index.html
[16] http://blackcat.brynmawr.edu/~nswoboda/prog-html.html

EXHIBIT 3

Invalidity of U.S. Patent 5,961,601 under 35 U.S.C. § 102 and/or § 103 by "Spinning the Web" by Yuval Fisher ("Spinning the Web")[1]

Spinning the Web, published in February 1996, qualifies as prior art to U.S. Patent No. 5,961,601 ("'601 patent") under 35 U.S.C. § 102(a) and/or (b) and anticipates and/or renders obvious one or more claims of the '601 patent.

As described in the following claim chart, one or more claims of the '601 patent are invalid as expressly and inherently anticipated by Spinning the Web, at least under Plaintiff's apparent infringement theory. Specifically, Defendant identifies relevant portions of the prior art in view of both (1) the Court's claim constructions and the plain meaning of the terms not construed, as well as (2) Plaintiff's infringement theory as set forth in Plaintiff's Final Infringement Contentions, even where inconsistent with the Court's claim constructions. Defendant does not agree that IBM has properly interpreted the Court's claim construction order nor that IBM has accurately identified functionality to meet each claim element. Nonetheless, to the extent understood, Defendant applies IBM's construction regarding such elements as indicated below.

In addition, to the extent that Spinning the Web is found not to anticipate, expressly or inherently, one or more of the asserted claims of the '601 patent, these claims are invalid as obvious in view of Spinning the Web alone or in combination with other prior art references, including but not limited to the prior art identified in Defendant's Final Invalidity Contentions and the prior art described in the claim charts attached thereto.

Claim Constructions

The Court has construed "continuation(s)" to mean "a new request which a client may send to a server, such as, for example, a hyperlink." Defendant has applied this understanding in its analysis in the chart below.

[1] The use of this reference or combinations of references as invalidating prior art under 35 U.S.C. §§ 102 and/or 103 may be based on Plaintiff's allegations of infringement. Defendant does not necessarily agree with the interpretations set forth in Plaintiff's infringement contentions and thus these invalidity contentions are not an admission that the accused products meet any particular claim element or infringe these claims. Moreover, nothing in these contentions should be interpreted as an acquiescence to or assertion of a particular claim construction by Defendant. In addition, nothing in these contentions should be interpreted as a position about whether any portion of the asserted claims is limiting or not. Further, by submitting these invalidity contentions, Defendant does not waive and hereby expressly reserves its right to raise other invalidity defenses, including but not limited to defenses under 35 U.S.C. § 112.

The Court has construed "all continuations in an output from said service" to mean "all new requests which a client may send to a server, such as, for example, a hyperlink, in an output from said service." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "recursively embedding the state information in all identified continuations" is "applying a process one or more times to each identified continuation to modify all identified continuations to include state information." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "conversation(s)" is "a sequence of communications between a client and server in which the server responds to each request with a set of continuations and the client always picks the next request from the set of continuations." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "filtering one of said hyperlinks and data output from said services according to a predetermined criteria" is "removing one of said hyperlinks and data output from said services according to criteria determined prior to removing." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "adding one of said hyperlinks and data to said output from said services according to a predetermined criteria" is "inserting one of said hyperlinks and data to said output from said services according to criteria determined prior to inserting." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "HTML" is "HyperText Markup Language." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "CGI program" is "Common Gateway Interface program." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "stateless protocol" is "a protocol where every request from a client to a server is treated independently of previous connections." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "client" is "a computer which issues commands to the server which performs the task associated with the command." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "state information" is "information about a conversation between a client and a server." Defendant has applied this understanding in its analysis in the chart below.

Claims of the '601 Patent	Spinning the Web
Claim 1	
[1.1] A computerized method for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client,	Spinning the Web discloses a method for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client. *See*, *e.g.*,: "The WWW is based on the Client/Server model of information exchange. This is much like dining in a restaurant in which the food is data. There must be a client; there must be a server; the client asks for things, such as menus and various entries; the server serves them to the client….Of course, both server and client are programs; the user interacts with the web using a client program, run on the local computer. The client is more complicated than the server, since it has to accept various formats of data, interpret them, and display them; the server must only send documents in response to requests from a client. The client program is run on the local computer. When a client wants to contact a server, it must know where one is. It must also know what to ask for. On the WWW the where and what are determined by a URL, the uniform resource locator." p.19-20. "http. HTTP is the 'native' protocol of the WWW, and hence http URLs are the most common." p. 21. "Even though servers log each interaction, the server program itself doesn't remember if the client has visited before. That is, each connection is stateless - it doesn't inherently contain information about previous or future requests." p. 44. *See generally* Chapter 2 "How the Web Works." "If it is important to maintain state - that is, to serve a document that depends on what happened before - then it is possible to do so by encoding the state in the requested URL (and the URLs returned by the server in a document)." p. 45.
[1.2] said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between	Spinning the Web discloses the services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation. *See*, *e.g.*,:

Claims of the '601 Patent	Spinning the Web
the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation, the method comprising the steps of:	"The WWW is based on the Client/Server model of information exchange. This is much like dining in a restaurant in which the food is data. There must be a client; there must be a server; the client asks for things, such as menus and various entries; the server serves them to the client….Of course, both server and client are programs; the user interacts with the web using a client program, run on the local computer. The client is more complicated than the server, since it has to accept various formats of data, interpret them, and display them; the server must only send documents in response to requests from a client. The client program is run on the local computer. When a client wants to contact a server, it must know where one is. It must also know what to ask for. On the WWW the where and what are determined by a URL, the uniform resource locator." p.19-20. "http. HTTP is the 'native' protocol of the WWW, and hence http URLs are the most common." p. 21. "Even though servers log each interaction, the server program itself doesn't remember if the client has visited before. That is, each connection is stateless - it doesn't inherently contain information about previous or future requests." p. 44. *See generally* Chapter 2 "How the Web Works." "Most WWW documents are written in HTML, the HyperText Markup Language. HTML tells the client how to display the document and where to seek related documents. It consists of tags that specify font size and style, images to include, text formatting commands, and URLs of other documents that can be retrieved from other WWW sites. Figure 1.2 shows a portion of a WWW document, both as it appears in the client window and the original text. It contains some text, an image, and some underlined text. The underlined text is an anchor or hyperlink, a link to another document. When the cursor is moved over a portion of underlined text and the mouse button is clicked (usually the left button, when there is more than one), the client program looks up the URL in the original text and retrieves that document. … This ability of text documents to reference other documents is called *hypertext*. . . . By simply pointing and clicking the mouse, it is very easy to move throughout the web." p. 23-25. "If it is important to maintain state - that is, to serve a document that depends on what happened before - then it is possible to do so by encoding the state in the requested URL (and the URLs

Claims of the '601 Patent	Spinning the Web
	returned by the server in a document)." p. 45.
[1.3] the client initiating the conversation with the server using the stateless protocol;	Spinning the Web discloses the client initiating the conversation with the server using the stateless protocol. *See*, *e.g.*,: "The WWW is based on the Client/Server model of information exchange. This is much like dining in a restaurant in which the food is data. There must be a client; there must be a server; the client asks for things, such as menus and various entries; the server serves them to the client….Of course, both server and client are programs; the user interacts with the web using a client program, run on the local computer. The client is more complicated than the server, since it has to accept various formats of data, interpret them, and display them; the server must only send documents in response to requests from a client. The client program is run on the local computer. When a client wants to contact a server, it must know where one is. It must also know what to ask for. On the WWW the where and what are determined by a URL, the uniform resource locator." p.19-20. "http. HTTP is the 'native' protocol of the WWW, and hence http URLs are the most common." p. 21. "Even though servers log each interaction, the server program itself doesn't remember if the client has visited before. That is, each connection is stateless - it doesn't inherently contain information about previous or future requests." p. 44. *See generally* Chapter 2 "How the Web Works." "Most WWW documents are written in HTML, the HyperText Markup Language. HTML tells the client how to display the document and where to seek related documents. It consists of tags that specify font size and style, images to include, text formatting commands, and URLs of other documents that can be retrieved from other WWW sites. Figure 1.2 shows a portion of a WWW document, both as it appears in the client window and the original text. It contains some text, an image, and some underlined text. The underlined text is an anchor or hyperlink, a link to another document. When the cursor is moved over a portion of underlined text and the mouse button is clicked (usually the left button, when there is more than one), the client program looks up the URL in the original text and retrieves that document. … This ability of text documents to reference other documents is called *hypertext*. . . . By simply pointing and clicking the mouse, it is very easy to

Claims of the '601 Patent	Spinning the Web
	move throughout the web." p. 23-25.
[1.4] detecting when the request for a service requires preservation of the state information;	Spinning the Web discloses detecting when the request for a service requires preservation of the state information. *See*, *e.g.*,: "If it is important to maintain state - that is, to serve a document that depends on what happened before - then it is possible to do so by encoding the state in the requested URL (and the URLs returned by the server in a document)." p. 45. "What these electronic merchants need is a way to add items to a 'shopping cart' that the purchaser can view and alter. This is not completely simple, because of the stateless server/client model of the WWW. That is, the server must remember who the client is, even though it is just serving documents to Internet hosts, without knowing the user on the other side. … Incidentally, schemes such as this are useful for more than just shopping baskets. See, for example, the game of adventure at [95]." p. 323.
[1.5] performing said service and identifying all continuations in an output from said service, in response to said step of detecting;	Spinning the Web discloses performing said service and identifying all continuations in an output from said service, in response to said step of detecting. *See*, *e.g.*,: "The state mechanism we describe here assigns a special number to each client. These are assigned when the client first connects. This number is included in all the URLs that the client sees after the initial connection, and hence all document requests to the server contain this identifying number in the submitted URL. This is good. It means that the server (that is, the CGI scripts used) can recognize to whom they are sending data by looking at their URL and extracting the unique number assigned to the specific client." p. 323. "To make life simple (both in this example and in the real world), we will create HTML templates that will be filtered for the addition of the client's identifier and then served. We use the special string

Claims of the '601 Patent	Spinning the Web

	### as a token that is replaced by the client's identifier before the template is served." p. 324.
	
	"We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY-STRING. It then filters the template, adding the identifier, and serves it up." p. 324.
	
	"When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier. The URL referring to see.pl also contains the name of a file, in this case store1.html, in which see.pl replaces occurrences of ### with the client identifier. This file is then served. Enter. Html looks like:
	
	`<HTML> <!-- enter.html: the store front -->`
	`<HEAD>`
	`<TITLE>Welcome to the Nothing Store</TITLE>`
	`</HEAD>`
	`<BODY>`
	`<H2>Welcome to the Nothing Store</H2>`
	`<A HREF="/cgi-bin/see.pl/store1.html?setup">`
	`Enter our Store</A>.`
	`<HR>`
	`<EM>Nothing Productions</EM>`
	`</BODY>`
	`</HTML>`
	
	…
	
	The template store1.html, rendered in Figure 9.4, looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- store1.html: The first store window --> <HEAD> <TITLE>Welcome to Store One</TITLE> </HEAD> <BODY> <H2>This is Store One</H2> <H3> The first item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item1?###>Add to Cart. <H3> The second item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item2?###>Add to Cart. <P> <H4><A HREF=/cgi-bin/cart.pl?###>See Cart Contents</H4> <P> See our <A HREF=/cgi-bin/see.pl/store2.html?###>other items</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> In store1.html there are two types of hyperlinks. One, near the bottom, calls see.pl in such a way that QUERY_STRING will be set to the client identifier (recall that the ### was replaced with it when see.pl was called the first time with QUERY_STRING=setup). In that call, PATH_INFO=/ store2. html, which means that following this link will cause see.pl to filter the contents of store2. html, again replacing the token ### with the client identifier. We do not show store2.html here, but it can be essentially identical to storel.html, with ones and twos exchanged." p. 325-26. "Moreover, all these URLs must reference CGI scripts (not static HTML pages), so that the client identifier can be propagated through whatever other pages are displayed." p. 327.

Claims of the '601 Patent	Spinning the Web
	See see.pl at p. 327-29; cart.pl at 330-32. *See generally* Section 9.5.10.
[1.6] recursively embedding the state information in all identified continuations; and	Spinning the Web discloses recursively embedding the state information in all identified continuations. *See*, *e.g.*,: "The state mechanism we describe here assigns a special number to each client. These are assigned when the client first connects. This number is included in all the URLs that the client sees after the initial connection, and hence all document requests to the server contain this identifying number in the submitted URL. This is good. It means that the server (that is, the CGI scripts used) can recognize to whom they are sending data by looking at their URL and extracting the unique number assigned to the specific client." p. 323. "To make life simple (both in this example and in the real world), we will create HTML templates that will be filtered for the addition of the client's identifier and then served. We use the special string ### as a token that is replaced by the client's identifier before the template is served." p. 324. "We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY-STRING. It then filters the template, adding the identifier, and serves it up." p. 324. "When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier. The URL referring to see.pl also contains the name of a file, in this case store1.html, in which see.pl replaces occurrences of ### with the client identifier. This file is then served. Enter. Html looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- enter.html: the store front --> <HEAD> <TITLE>Welcome to the Nothing Store</TITLE> </HEAD> <BODY> <H2>Welcome to the Nothing Store</H2> <A HREF="/cgi-bin/see.pl/store1.html?setup"> Enter our Store</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> … The template store1.html, rendered in Figure 9.4, looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- store1.html: The first store window --> <HEAD> <TITLE>Welcome to Store One</TITLE> </HEAD> <BODY> <H2>This is Store One</H2> <H3> The first item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item1?###>Add to Cart. <H3> The second item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item2?###>Add to Cart. <P> <H4><A HREF=/cgi-bin/cart.pl?###>See Cart Contents</H4> <P> See our <A HREF=/cgi-bin/see.pl/store2.html?###>other items</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> In store1.html there are two types of hyperlinks. One, near the bottom, calls see.pl in such a way that QUERY_STRING will be set to the client identifier (recall that the ### was replaced with it when see.pl was called the first time with QUERY_STRING=setup). In that call, PATH_INFO=/ store2. html, which means that following this link will cause see.pl to filter the contents of store2. html, again replacing the token ### with the client identifier. We do not show store2.html here, but it can be essentially identical to storel.html, with ones and twos exchanged." p. 325-26. "Moreover, all these URLs must reference CGI scripts (not static HTML pages), so that the client identifier can be propagated through whatever other pages are displayed." p. 327.

Claims of the '601 Patent	Spinning the Web
	See see.pl at p. 327-29; cart.pl at 330-32' *see generally* Section 9.5.10.
[1.7] communicating the output to the client, in response to said step of embedding, wherein the state information is preserved and provided to all services for the duration of the conversation.	Spinning the Web discloses communicating the output to the client, in response to said step of embedding, wherein the state information is preserved and provided to all services for the duration of the conversation. *See*, *e.g.*,: "The state mechanism we describe here assigns a special number to each client. These are assigned when the client first connects. This number is included in all the URLs that the client sees after the initial connection, and hence all document requests to the server contain this identifying number in the submitted URL. This is good. It means that the server (that is, the CGI scripts used) can recognize to whom they are sending data by looking at their URL and extracting the unique number assigned to the specific client." p. 323. "We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY-STRING. It then filters the template, adding the identifier, and serves it up." p. 324. *See* see.pl at p. 327-29; cart.pl at 330-32' *see generally* Section 9.5.10.
Claim 2	
[2.1] The method of claim 1, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding.	Spinning the Web discloses the method of claim 1, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding. *See*, *e.g.*,: "We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY-STRING. It then filters the template, adding the identifier, and serves it up." p. 324. *See* see.pl at p. 327-29; cart.pl at 330-32' *see generally* Section 9.5.10.
Claim 3	
[3.1] The method of claim 2,	Spinning the Web discloses the method of claim 2, further comprising the step of storing at least part

Claims of the '601 Patent	Spinning the Web

further comprising the step of storing at least part of the state information in a memory coupled to the server and	of the state information in a memory coupled to the server, such as in environment variables accessible by a CGI script. *See*, *e.g.*,:
	
	See see.pl at p. 327-29; cart.pl at 330-32' *see generally* Section 9.5.10.
	
	"We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY_STRING. It then filters the template, adding the identifier, and serves it up." p. 324.
	
	"When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier. The URL referring to see.pl also contains the name of a file, in this case store1.html, in which see.pl replaces occurrences of ### with the client identifier. This file is then served. Enter.html looks like:
	
	`<HTML> <!-- enter.html: the store front -->`
	`<HEAD>`
	`<TITLE>Welcome to the Nothing Store</TITLE>`
	`</HEAD>`
	`<BODY>`
	`<H2>Welcome to the Nothing Store</H2>`
	`<A HREF="/cgi-bin/see.pl/store1.html?setup">`
	`Enter our Store</A>.`
	`<HR>`
	`<EM>Nothing Productions</EM>`
	`</BODY>`
	`</HTML>`
	
	…
	
	The template store1.html, rendered in Figure 9.4, looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- store1.html: The first store window --> <HEAD> <TITLE>Welcome to Store One</TITLE> </HEAD> <BODY> <H2>This is Store One</H2> <H3> The first item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item1?###>Add to Cart. <H3> The second item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item2?###>Add to Cart. <P> <H4><A HREF=/cgi-bin/cart.pl?###>See Cart Contents</H4> <P> See our <A HREF=/cgi-bin/see.pl/store2.html?###>other items</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> In store1.html there are two types of hyperlinks. One, near the bottom, calls see.pl in such a way that QUERY_STRING will be set to the client identifier (recall that the ### was replaced with it when see.pl was called the first time with QUERY_STRING=setup). In that call, PATH_INFO=/ store2. html, which means that following this link will cause see.pl to filter the contents of store2. html, again replacing the token ### with the client identifier. We do not show store2.html here, but it can be essentially identical to storel.html, with ones and twos exchanged." p. 325-26. "Moreover, all these URLs must reference CGI scripts (not static HTML pages), so that the client identifier can be propagated through whatever other pages are displayed." p. 327.

Claims of the '601 Patent	Spinning the Web
	See see.pl at p. 327-29; cart.pl at 330-32. *See generally* Section 9.5.10.
[3.2] wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations.	Spinning the Web discloses wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations. *See*, *e.g.*,: Spinning the Web discloses the use of a hash structure to hold state information, e.g. the environment variables which hold QUERY_STRING and other information. One of ordinary skill in the art would know that *See, e.g.*, p. 328 at line 19 (extracting QUERY_STRING from the hash holding environment variables); p. 331 at line 25 (same). One of ordinary skill in the art reading Spinning the Web would know that such variables would be stored in the memory of the server executing the Perl script. *See* see.pl at p. 327-29; cart.pl at 330-32; *see generally* Section 9.5.10. "We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY_STRING. It then filters the template, adding the identifier, and serves it up." p. 324. "When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier. The URL referring to see.pl also contains the name of a file, in this case store1.html, in which see.pl replaces occurrences of ### with the client identifier. This file is then served. Enter.html looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- enter.html: the store front --> <HEAD> <TITLE>Welcome to the Nothing Store</TITLE> </HEAD> <BODY> <H2>Welcome to the Nothing Store</H2> <A HREF="/cgi-bin/see.pl/store1.html?setup"> Enter our Store</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> … The template store1.html, rendered in Figure 9.4, looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- store1.html: The first store window --> <HEAD> <TITLE>Welcome to Store One</TITLE> </HEAD> <BODY> <H2>This is Store One</H2> <H3> The first item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item1?###>Add to Cart. <H3> The second item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item2?###>Add to Cart. <P> <H4><A HREF=/cgi-bin/cart.pl?###>See Cart Contents</H4> <P> See our <A HREF=/cgi-bin/see.pl/store2.html?###>other items</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> In store1.html there are two types of hyperlinks. One, near the bottom, calls see.pl in such a way that QUERY_STRING will be set to the client identifier (recall that the ### was replaced with it when see.pl was called the first time with QUERY_STRING=setup). In that call, PATH_INFO=/ store2. html, which means that following this link will cause see.pl to filter the contents of store2. html, again replacing the token ### with the client identifier. We do not show store2.html here, but it can be essentially identical to storel.html, with ones and twos exchanged." p. 325-26. "Moreover, all these URLs must reference CGI scripts (not static HTML pages), so that the client identifier can be propagated through whatever other pages are displayed." p. 327.

Claims of the '601 Patent	Spinning the Web
	In addition, to the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Spinning the Web to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 4	
[4.1] The method of claim 1, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client.	Spinning the Web discloses the method of claim 1, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client. *See*, *e.g.*,: Spinning the Web discloses the use of Java and JavaScript as client-side dynamically-downloaded computer program code. "Java is a secure language that will probably play a role on the WWW, due to Netscape's commitment to support it. It allows the server to send a program to be executed by the client. Such a program can, for example, fetch information, interact with the user locally, and display animations. An introduction to Java can be found in Chapter 12." p. 28. *See* Williams. "JavaScript is a language built into Netscape Navigator (Versions 2.0 beta and later). It is likely to be built into many browsers, since Microsoft has endorsed the Java language. It is based on, but simpler than, Java. Unlike Java, JavaScript doesn't have rigidly typed variables, and it allows functions rather than insisting on encapsulated methods. Classes cannot be created, but a large number of classes and objects based on the document's HTML are created when the page is loaded. In this way it is possible to access object information for the HTML in the document. For example, it is possible to read form information, react to mouse clicks, modify input fields, and so on." p. 414. Spinning the Web also discloses the extraction of information from a URL via Javascript. *See* Section 12.8.3 at p. 429-30. Spinning the Web further discloses accessing all links in a current HTML document. *See* p. 432. It would have been obvious to one ordinary skill in the art reading Spinning the Web to combine the shopping cart example in section 9.5.10 with these disclosures to embed the client identifier in links

Claims of the '601 Patent	Spinning the Web
	found on the received HTML page rather than doing so on the server side. In addition, to the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Spinning the Web to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 5	
[5.1] The method of claim 4, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information.	Spinning the Web discloses the method of claim 4, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information. *See*, *e.g.*,: Spinning the Web discloses the use of Java and JavaScript as client-side dynamically-downloaded computer program code. "Java is a secure language that will probably play a role on the WWW, due to Netscape's commitment to support it. It allows the server to send a program to be executed by the client. Such a program can, for example, fetch information, interact with the user locally, and display animations. An introduction to Java can be found in Chapter 12." p. 28. *See* Williams. "JavaScript is a language built into Netscape Navigator (Versions 2.0 beta and later). It is likely to be built into many browsers, since Microsoft has endorsed the Java language. It is based on, but simpler than, Java. Unlike Java, JavaScript doesn't have rigidly typed variables, and it allows functions rather than insisting on encapsulated methods. Classes cannot be created, but a large number of classes and objects based on the document's HTML are created when the page is loaded. In this way it is possible to access object information for the HTML in the document. For example, it is possible to read form information, react to mouse clicks, modify input fields, and so on." p. 414. Spinning the Web also discloses the extraction of information from a URL via Javascript. *See* Section 12.8.3 at p. 429-30. Spinning the Web further discloses accessing all links in a current HTML document. *See* p. 432. It would have been obvious to one ordinary skill in the art reading Spinning the Web to combine the

Claims of the '601 Patent	Spinning the Web
	shopping cart example in section 9.5.10 with these disclosures to embed the client identifier in links found on the received HTML page rather than doing so on the server side. In addition, to the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Spinning the Web to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 6	
[6.1] The method of claim 1, further comprising the steps of: the client selecting a second continuation from said all identified continuations with embedded state information; and	Spinning the Web discloses the method of claim 1, further comprising the steps of: the client selecting a second continuation from said all identified continuations with embedded state information. *See*, *e.g.*,: "The state mechanism we describe here assigns a special number to each client. These are assigned when the client first connects. This number is included in all the URLs that the client sees after the initial connection, and hence all document requests to the server contain this identifying number in the submitted URL. This is good. It means that the server (that is, the CGI scripts used) can recognize to whom they are sending data by looking at their URL and extracting the unique number assigned to the specific client." p. 323. "To make life simple (both in this example and in the real world), we will create HTML templates that will be filtered for the addition of the client's identifier and then served. We use the special string ### as a token that is replaced by the client's identifier before the template is served." p. 324. "We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY_STRING. It then filters the template, adding the identifier, and serves it up." p. 324. "When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier. The URL referring to see.pl also contains the name of a file, in this case store1.html, in which see.pl replaces occurrences of ### with the client identifier. This file is then served. Enter.html looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- enter.html: the store front --> <HEAD> <TITLE>Welcome to the Nothing Store</TITLE> </HEAD> <BODY> <H2>Welcome to the Nothing Store</H2> <A HREF="/cgi-bin/see.pl/store1.html?setup"> Enter our Store</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> … The template store1.html, rendered in Figure 9.4, looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- store1.html: The first store window --> <HEAD> <TITLE>Welcome to Store One</TITLE> </HEAD> <BODY> <H2>This is Store One</H2> <H3> The first item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item1?###>Add to Cart. <H3> The second item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item2?###>Add to Cart. <P> <H4><A HREF=/cgi-bin/cart.pl?###>See Cart Contents</H4> <P> See our <A HREF=/cgi-bin/see.pl/store2.html?###>other items</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> In store1.html there are two types of hyperlinks. One, near the bottom, calls see.pl in such a way that QUERY_STRING will be set to the client identifier (recall that the ### was replaced with it when see.pl was called the first time with QUERY_STRING=setup). In that call, PATH_INFO=/ store2.html, which means that following this link will cause see.pl to filter the contents of store2.html, again replacing the token ### with the client identifier. We do not show store2.html here, but it can be essentially identical to storel.html, with ones and twos exchanged." p. 325-26. "Moreover, all these URLs must reference CGI scripts (not static HTML pages), so that the client identifier can be propagated through whatever other pages are displayed." p. 327.

Claims of the '601 Patent	Spinning the Web
[6.2] restoring the state information from said second continuation and invoking an associated second service with restored state information;	Spinning the Web discloses restoring the state information from said second continuation and invoking an associated second service with restored state information. *See*, *e.g.*,: Spinning the Web discloses a second service, the cart CGI script, which is invoked by a user after the see CGI script restores the client identifier and passes it to a continuation calling the cart CGI service. "What these electronic merchants need is a way to add items to a 'shopping cart' that the purchaser can view and alter. This is not completely simple, because of the stateless server/client model of the WWW. That is, the server must remember who the client is, even though it is just serving documents to Internet hosts, without knowing the user on the other side. … Incidentally, schemes such as this are useful for more than just shopping baskets. See, for example, the game of adventure at [95]." p. 323. "To make life simple (both in this example and in the real world), we will create HTML templates that will be filtered for the addition of the client's identifier and then served. We use the special string ### as a token that is replaced by the client's identifier before the template is served." p. 324. "We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY_STRING. It then filters the template, adding the identifier, and serves it up." p. 324. "When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier. The URL referring to see.pl also contains the name of a file, in this case store1.html, in which see.pl replaces occurrences of ### with the client identifier. This file is then served. Enter.html looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- enter.html: the store front --> <HEAD> <TITLE>Welcome to the Nothing Store</TITLE> </HEAD> <BODY> <H2>Welcome to the Nothing Store</H2> <A HREF="/cgi-bin/see.pl/store1.html?setup"> Enter our Store</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> … The template store1.html, rendered in Figure 9.4, looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- store1.html: The first store window --> <HEAD> <TITLE>Welcome to Store One</TITLE> </HEAD> <BODY> <H2>This is Store One</H2> <H3> The first item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item1?###>Add to Cart. <H3> The second item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item2?###>Add to Cart. <P> <H4><A HREF=/cgi-bin/cart.pl?###>See Cart Contents</H4> <P> See our <A HREF=/cgi-bin/see.pl/store2.html?###>other items</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> In store1.html there are two types of hyperlinks. One, near the bottom, calls see.pl in such a way that QUERY_STRING will be set to the client identifier (recall that the ### was replaced with it when see.pl was called the first time with QUERY_STRING=setup). In that call, PATH_INFO=/ store2.html, which means that following this link will cause see.pl to filter the contents of store2.html, again replacing the token ### with the client identifier. We do not show store2.html here, but it can be essentially identical to storel.html, with ones and twos exchanged." p. 325-26. "Moreover, all these URLs must reference CGI scripts (not static HTML pages), so that the client identifier can be propagated through whatever other pages are displayed." p. 327.

Claims of the '601 Patent	Spinning the Web
[6.3] recursively identifying and embedding the state information in all continuations associated with an output from said second service.	Spinning the Web discloses recursively identifying and embedding the state information in all continuations associated with an output from said second service. *See*, *e.g.*,: "The second type of hyperlink in store.html calls the CGI script cart.pl. This script also receives the client identifier in QUERY_STRING, and it receives a string, e.g., store1_item1 in PATH_INFO, that tells it which item was put in the shopping cart." *See* see.pl at p. 327-29; cart.pl at 330-32 at ll. 63-65 (embedding the client id in each hyperlink on the shopping cart page); *see also* 333-34 (result of calling cart.pl is that each hyperlink includes the client id).
Claim 7	
[7.1] The method of claim 1, further comprising the step of correlating the state information to a specific conversation.	Spinning the Web discloses the method of claim 1, further comprising the step of correlating the state information to a specific conversation. *See*, *e.g.*,: Spinning the Web discloses that once a conversation begins, all URLs should include the client id. A new conversation may begin by going to the entrance page to set a new client id. "The state mechanism we describe here assigns a special number to each client. These are assigned when the client first connects. This number is included in all the URLs that the client sees after the initial connection, and hence all document requests to the server contain this identifying number in the submitted URL. This is good. It means that the server (that is, the CGI scripts used) can recognize to whom they are sending data by looking at their URL and extracting the unique number assigned to

Claims of the '601 Patent	Spinning the Web
	the specific client." p. 323. "The topmost page is an entrance, enter. htrnl from which we call the CGI script see.pl with a special value, setup, that will be put in QUERY-STRING. When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier." p. 324-25.
Claim 8	
[8.1] The method of claim 1, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	Spinning the Web discloses wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs. *See*, *e.g.*,: "The WWW is based on the Client/Server model of information exchange. This is much like dining in a restaurant in which the food is data. There must be a client; there must be a server; the client asks for things, such as menus and various entries; the server serves them to the client….Of course, both server and client are programs; the user interacts with the web using a client program, run on the local computer. The client is more complicated than the server, since it has to accept various formats of data, interpret them, and display them; the server must only send documents in response to requests from a client. The client program is run on the local computer. When a client wants to contact a server, it must know where one is. It must also know what to ask for. On the WWW the where and what are determined by a URL, the uniform resource locator." p.19-20. "http. HTTP is the 'native' protocol of the WWW, and hence http URLs are the most common." p. 21. "Most WWW documents are written in HTML, the HyperText Markup Language. HTML tells the client how to display the document and where to seek related documents. It consists of tags that specify font size and style, images to include, text formatting commands, and URLs of other documents that can be retrieved from other WWW sites. Figure 1.2 shows a portion of a WWW document, both as it appears in the client window and the original text. It contains some text, an image, and some underlined text. The underlined text is an anchor or hyperlink, a link to another document. When the cursor is moved over a portion of underlined text and the mouse button is

Claims of the '601 Patent	Spinning the Web
	clicked (usually the left button, when there is more than one), the client program looks up the URL in the original text and retrieves that document. … This ability of text documents to reference other documents is called *hypertext*. . . . By simply pointing and clicking the mouse, it is very easy to move throughout the web." p. 23-25. "CGI Scripts. The functionality of the server can be extended by writing scripts that can perform many types of functions, for example, searching a database or forwarding e-mail. The Common Gateway Interface (CGI) specifies how the server and script interface; it is the topic of Chapter 9." p. 34 "Even though servers log each interaction, the server program itself doesn't remember if the client has visited before. That is, each connection is stateless - it doesn't inherently contain information about previous or future requests." p. 44. *See generally* Chapter 2 "How the Web Works." "If it is important to maintain state - that is, to serve a document that depends on what happened before - then it is possible to do so by encoding the state in the requested URL (and the URLs returned by the server in a document)." p. 45. *See* see.pl at p. 327-29; cart.pl at 330-32. *See generally* Section 9.5.10.
Claim 9	
[9.1] The method of claim 8, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	Spinning the Web discloses the method of claim 8, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria. *See*, *e.g.*,: Spinning the Web discloses filtering the data output by identifying only items that include a special placeholder to be replaced with a client id in order to add it to a hyperlink. "The state mechanism we describe here assigns a special number to each client. These are assigned when the client first connects. This number is included in all the URLs that the client sees after the initial connection, and hence all document requests to the server contain this identifying number in

Claims of the '601 Patent	Spinning the Web
	the submitted URL. This is good. It means that the server (that is, the CGI scripts used) can recognize to whom they are sending data by looking at their URL and extracting the unique number assigned to the specific client." p. 323. "To make life simple (both in this example and in the real world), we will create HTML templates that will be filtered for the addition of the client's identifier and then served. We use the special string ### as a token that is replaced by the client's identifier before the template is served." p. 324. "We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY_STRING. It then filters the template, adding the identifier, and serves it up." p. 324. *See* see.pl at p. 327-29; cart.pl at 330-32; *see generally* Section 9.5.10. In addition, to the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Spinning the Web to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 10	
[10.1] The method of claim 8, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	Spinning the Web discloses the method of claim 8, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria. *See, e.g.*,: Spinning the Web discloses filtering the data output by identifying only items that include a special placeholder to be replaced with a client id in order to add it to a hyperlink. "The state mechanism we describe here assigns a special number to each client. These are assigned when the client first connects. This number is included in all the URLs that the client sees after the initial connection, and hence all document requests to the server contain this identifying number in the submitted URL. This is good. It means that the server (that is, the CGI scripts used) can recognize to whom they are sending data by looking at their URL and extracting the unique number assigned to

Claims of the '601 Patent	Spinning the Web
	the specific client." p. 323. "To make life simple (both in this example and in the real world), we will create HTML templates that will be filtered for the addition of the client's identifier and then served. We use the special string ### as a token that is replaced by the client's identifier before the template is served." p. 324. "We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY_STRING. It then filters the template, adding the identifier, and serves it up." p. 324. *See* see.pl at p. 327-29; cart.pl at 330-32; *see generally* Section 9.5.10. In addition, to the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Spinning the Web to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 11	
[11.1] The method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments.	Spinning the Web discloses the method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments. *See*, *e.g.*,: Spinning the Web discloses filtering the data output by identifying only items that include a special placeholder to be replaced with a client id in order to add it to a hyperlink. "The state mechanism we describe here assigns a special number to each client. These are assigned when the client first connects. This number is included in all the URLs that the client sees after the initial connection, and hence all document requests to the server contain this identifying number in the submitted URL. This is good. It means that the server (that is, the CGI scripts used) can recognize to whom they are sending data by looking at their URL and extracting the unique number assigned to

Claims of the '601 Patent	Spinning the Web

	the specific client." p. 323.
	
	"To make life simple (both in this example and in the real world), we will create HTML templates that will be filtered for the addition of the client's identifier and then served. We use the special string ### as a token that is replaced by the client's identifier before the template is served." p. 324.
	
	"We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY_STRING. It then filters the template, adding the identifier, and serves it up." p. 324.
	
	"When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier. The URL referring to see.pl also contains the name of a file, in this case store1.html, in which see.pl replaces occurrences of ### with the client identifier. This file is then served. Enter.html looks like:
	
	`<HTML> <!-- enter.html: the store front -->`
	`<HEAD>`
	`<TITLE>Welcome to the Nothing Store</TITLE>`
	`</HEAD>`
	`<BODY>`
	`<H2>Welcome to the Nothing Store</H2>`
	`<A HREF="/cgi-bin/see.pl/store1.html?setup">`
	`Enter our Store</A>.`
	`<HR>`
	`<EM>Nothing Productions</EM>`
	`</BODY>`
	`</HTML>`
	
	…
	
	The template store1.html, rendered in Figure 9.4, looks like:

Claims of the '601 Patent	Spinning the Web

```
<HTML> <!-- store1.html: The first store window -->
<HEAD>
 <TITLE>Welcome to Store One</TITLE>
</HEAD>
<BODY>
 <H2>This is Store One</H2>
 <H3> The first item is Nothing </H3>
 <A HREF=/cgi-bin/cart.pl/store1_item1?###>Add to Cart</a>.
 <H3> The second item is Nothing </H3>
 <A HREF=/cgi-bin/cart.pl/store1_item2?###>Add to Cart</a>.
 <P>
 <H4><A HREF=/cgi-bin/cart.pl?###>See Cart Contents</a></H4>
 <P>
 See our
 <A HREF=/cgi-bin/see.pl/store2.html?###>other items</A>.
 <HR>
 <EM>Nothing Productions</EM>
</BODY>
</HTML>
```

In store1.html there are two types of hyperlinks. One, near the bottom, calls see.pl in such a way that QUERY_STRING will be set to the client identifier (recall that the ### was replaced with it when see.pl was called the first time with QUERY_STRING=setup). In that call, PATH_INFO=/ store2.html, which means that following this link will cause see.pl to filter the contents of store2.html, again replacing the token ### with the client identifier. We do not show store2.html here, but it can be essentially identical to storel.html, with ones and twos exchanged." p. 325-26.

"Moreover, all these URLs must reference CGI scripts (not static HTML pages), so that the client identifier can be propagated through whatever other pages are displayed." p. 327.

Claims of the '601 Patent	Spinning the Web
	See see.pl at p. 327-29; cart.pl at 330-32. *See generally* Section 9.5.10.
Claim 12	
[12.1] The method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program.	Spinning the Web discloses the method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program. *See*, *e.g.*,: "What these electronic merchants need is a way to add items to a 'shopping cart' that the purchaser can view and alter. This is not completely simple, because of the stateless server/client model of the WWW. That is, the server must remember who the client is, even though it is just serving documents to Internet hosts, without knowing the user on the other side. … Incidentally, schemes such as this are useful for more than just shopping baskets. See, for example, the game of adventure at [95]." p. 323. "To make life simple (both in this example and in the real world), we will create HTML templates that will be filtered for the addition of the client's identifier and then served. We use the special string ### as a token that is replaced by the client's identifier before the template is served." p. 324. "We will require two CGI scripts. The first, see.pl, serves a document that includes a client's identifier. It finds the template to be served in PATH_INFO, and it finds the client's identifier in QUERY_STRING. It then filters the template, adding the identifier, and serves it up." p. 324. "When see.pl sees that QUERY_STRING has the value setup, it knows a new person has come to the store and it creates a new client identifier. The URL referring to see.pl also contains the name of a file, in this case store1.html, in which see.pl replaces occurrences of ### with the client identifier. This file is then served. Enter.html looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- enter.html: the store front --> <HEAD> <TITLE>Welcome to the Nothing Store</TITLE> </HEAD> <BODY> <H2>Welcome to the Nothing Store</H2> <A HREF="/cgi-bin/see.pl/store1.html?setup"> Enter our Store</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> … The template store1.html, rendered in Figure 9.4, looks like:

Claims of the '601 Patent	Spinning the Web
	<HTML> <!-- store1.html: The first store window --> <HEAD> <TITLE>Welcome to Store One</TITLE> </HEAD> <BODY> <H2>This is Store One</H2> <H3> The first item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item1?###>Add to Cart. <H3> The second item is Nothing </H3> <A HREF=/cgi-bin/cart.pl/store1_item2?###>Add to Cart. <P> <H4><A HREF=/cgi-bin/cart.pl?###>See Cart Contents</H4> <P> See our <A HREF=/cgi-bin/see.pl/store2.html?###>other items</A>. <HR> <EM>Nothing Productions</EM> </BODY> </HTML> In store1.html there are two types of hyperlinks. One, near the bottom, calls see.pl in such a way that QUERY_STRING will be set to the client identifier (recall that the ### was replaced with it when see.pl was called the first time with QUERY_STRING=setup). In that call, PATH_INFO=/ store2.html, which means that following this link will cause see.pl to filter the contents of store2.html, again replacing the token ### with the client identifier. We do not show store2.html here, but it can be essentially identical to storel.html, with ones and twos exchanged." p. 325-26. "Moreover, all these URLs must reference CGI scripts (not static HTML pages), so that the client identifier can be propagated through whatever other pages are displayed." p. 327. "The second type of hyperlink in store.html calls the CGI script cart.pl. This script also receives the

Claims of the '601 Patent	Spinning the Web
	client identifier in QUERY_STRING, and it receives a string, e.g., store1_item1 in PATH_INFO, that tells it which item was put in the shopping cart." *See* see.pl at p. 327-29; cart.pl at 330-32 at ll. 63-65 (embedding the client id in each hyperlink on the shopping cart page); *see also* 333-34 (result of calling cart.pl is that each hyperlink includes the client id). *See* see.pl at p. 327-29; cart.pl at 330-32; *see generally* Section 9.5.10.
Claim 51	
[51.1] A computerized method for preserving state information in a conversation via a stateless protocol between a client adapted to request services from one or more servers, the method comprising the steps of:	Spinning the Web discloses a computerized method for preserving state information in a conversation via a stateless protocol between a client adapted to request services from one or more servers. *See* claim 1.1.
[51.2] receiving a service request including state information, via the stateless protocol;	Spinning the Web discloses receiving a service request including state information, via the stateless protocol. *See* claim 1.3.
[51.3] identifying all continuations in an output from said service and recursively embedding the state information in all identified continuations, in response to said request; and	Spinning the Web discloses identifying all continuations in an output from said service and recursively embedding the state information in all identified continuations, in response to said request. *See* claims 1.5-1.6.
[51.4] communicating a response including the continuations and embedded state information, wherein	Spinning the Web discloses communicating a response including the continuations and embedded state information, wherein the continuations enable another service request and one of the continuations must be invoked to continue the conversation. *See* claim 1.7.

Claims of the '601 Patent	Spinning the Web
the continuations enable another service request and one of the continuations must be invoked to continue the conversation.	
Claim 52	
[52.1] The method of claim 51, wherein said embedding is performed by a server and said step of communicating is in response to said embedding step.	Spinning the Web discloses the method of claim 51, wherein said embedding is performed by a server and said step of communicating is in response to said embedding step. *See* claim 2.1.
Claim 53	
[53.1] The method of claim 52, further comprising the step of storing at least part of the state information in a memory coupled to the server and	Spinning the Web discloses the method of claim 52, further comprising the step of storing at least part of the state information in a memory coupled to the server. *See* claim 3.1.
[53.2] wherein embedding step includes embedding an index representing said part of the state information in said all identified continuations.	Spinning the Web discloses wherein embedding step includes embedding an index representing said part of the state information in said all identified continuations. *See* claim 3.2.
Claim 54	
[54.1] The method of claim 51, further comprising the step of dynamically downloading computer program code to the client to perform said embedding	Spinning the Web discloses the method of claim 51, further comprising the step of dynamically downloading computer program code to the client to perform said embedding step, in response to said step of communicating the output to the client. *See* claim 4.1.

Claims of the '601 Patent	Spinning the Web
step, in response to said step of communicating the output to the client.	
Claim 55	
[55.1] The method of claim 54, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said embedding step includes embedding an index representing said part of the state information.	Spinning the Web discloses the method of claim 54, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said embedding step includes embedding an index representing said part of the state information. *See* claim 5.1.
Claim 56	
[56.1] The method of claim 51, further comprising the steps of: receiving a second request associated with a second continuation from said all identified continuations with embedded state information; and restoring the state information from said second continuation and invoking an associated second service with restored state information; recursively identifying and embedding the state	Spinning the Web discloses the method of claim 51, further comprising the steps of: receiving a second request associated with a second continuation from said all identified continuations with embedded state information; and restoring the state information from said second continuation and invoking an associated second service with restored state information; recursively identifying and embedding the state information in all continuations associated with an output from said second service. *See* claims 6.1-6.3.

Claims of the '601 Patent	Spinning the Web
information in all continuations associated with an output from said second service.	
Claim 57	
[57.1] The method of claim 51, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	Spinning the Web discloses The method of claim 51, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs. *See* claim 8.1.
Claim 58	
[58.1] The method of claim 57, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	Spinning the Web discloses the method of claim 57, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria. *See* claim 9.1.
Claim 59	
[59.1] The method of claim 57, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	Spinning the Web discloses the method of claim 57, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria. *See* claim 10.1.

EXHIBIT 4

Invalidity of U.S. Patent 5,961,601 under 35 U.S.C. § 102 and/or § 103 by the Amazon.com Website in 1995 ("Amazon Prior Art System")[1]

The Amazon.com website, available and in public use in 1995, qualifies as prior art to U.S. Patent No. 5,961,601 ("'601 patent") under 35 U.S.C. § 102(a), (b) or (g) and anticipates and/or renders obvious one or more claims of the '601 patent.

As described in the following claim chart, one or more claims of the '601 patent are invalid as expressly and inherently anticipated by the Amazon Prior Art System, at least under Plaintiff's apparent infringement theory. Specifically, Defendant identifies relevant portions of the prior art in view of both (1) the Court's claim constructions and the plain meaning of the terms not construed, as well as (2) Plaintiff's infringement theory as set forth in Plaintiff's Final Infringement Contentions, even where inconsistent with the Court's claim constructions. Defendant does not agree that IBM has properly interpreted the Court's claim construction order nor that IBM has accurately identified functionality to meet each claim element. Nonetheless, to the extent understood, Defendant applies IBM's construction regarding such elements as indicated below.

In addition, to the extent that the Amazon Prior Art System is found not to anticipate, expressly or inherently, one or more of the asserted claims of the '601 patent, these claims are invalid as obvious in view of the Amazon Prior Art System alone or in combination with other prior art references, including but not limited to the prior art identified in Defendant's Final Invalidity Contentions and the prior art described in the claim charts attached thereto.

Claim Constructions

The Court has construed "continuation(s)" to mean "a new request which a client may send to a server, such as, for example, a hyperlink." Defendant has applied this understanding in its analysis in the chart below.

[1] The use of this reference or combinations of references as invalidating prior art under 35 U.S.C. §§ 102 and/or 103 may be based on Plaintiff's allegations of infringement. Defendant does not necessarily agree with the interpretations set forth in Plaintiff's infringement contentions and thus these invalidity contentions are not an admission that the accused products meet any particular claim element or infringe these claims. Moreover, nothing in these contentions should be interpreted as an acquiescence to or assertion of a particular claim construction by Defendant. In addition, nothing in these contentions should be interpreted as a position about whether any portion of the asserted claims is limiting or not. Further, by submitting these invalidity contentions, Defendant does not waive and hereby expressly reserves its right to raise other invalidity defenses, including but not limited to defenses under 35 U.S.C. § 112.

The Court has construed "all continuations in an output from said service" to mean "all new requests which a client may send to a server, such as, for example, a hyperlink, in an output from said service." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "recursively embedding the state information in all identified continuations" is "applying a process one or more times to each identified continuation to modify all identified continuations to include state information." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "conversation(s)" is "a sequence of communications between a client and server in which the server responds to each request with a set of continuations and the client always picks the next request from the set of continuations." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "filtering one of said hyperlinks and data output from said services according to a predetermined criteria" is "removing one of said hyperlinks and data output from said services according to criteria determined prior to removing." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "adding one of said hyperlinks and data to said output from said services according to a predetermined criteria" is "inserting one of said hyperlinks and data to said output from said services according to criteria determined prior to inserting." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "HTML" is "HyperText Markup Language." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "CGI program" is "Common Gateway Interface program." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "stateless protocol" is "a protocol where every request from a client to a server is treated independently of previous connections." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "client" is "a computer which issues commands to the server which performs the task associated with the command." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "state information" is "information about a conversation between a client and a server." Defendant has applied this understanding in its analysis in the chart below.

Claims of the '601 Patent	The Amazon Prior Art System
Claim 1	
[1.1] A computerized method for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client,	The Amazon Prior Art System discloses a method for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client. *See*, *e.g.*,: In the Amazon Prior Art System, the customer used a web browser running locally on the customer's computer to interact with services running on Amazon's computers. The customer's web browser would communicate over the Internet (using HTTP) with Amazon's web server. Amazon's web server would receive HTTP requests from the customer's computer. The web server would communicate with the Amazon obidos system to process requests from the customer and to generate HTML pages. As part of processing requests and generating web pages, obidos would communicate with databases that stored information. Obidos would send HTML pages to the web server. The web server would communicate the HTML pages to the customer's computer over the Internet (using HTTP).
[1.2] said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for	The Amazon Prior Art System discloses the services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation. *See*, *e.g.*,: When a customer was interacting with the Amazon Prior Art System, the obidos system assigned the customer a session identifier. Obidos included the session identifier in the HTML page sent to the user in one of two ways: (1) a hidden field in a form; or (2) as part of the path in a HREF tag. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/uid.c: assign_uid(). The session identifier was a key to an item in the sessions table in a database, which stored information about the state of the user's session. *See, e.g.*, AllFilesJune1995 at src/oracle-access/libacb/session.c: session_save(). The

Claims of the '601 Patent	The Amazon Prior Art System

said services and wherein the client must invoke one of the continuations to continue the conversation, the method comprising the steps of:	session identifier allowed the obidos system to maintain a user's state between HTTP responses. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main().
	
	The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*, AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[2] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as

[2] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System
	variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp: cat_subst(stdout, webfile("user-review.html"), uid, isbn, title, authbuff, 0); AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id. Based partially on the HTML generated during the call to `cat_subst()`, the Amazon Prior Art System generated a web page that was sent to the customer. Included in the generated web page was the customer's Session ID. As an example, when the customer confirmed the order, the customer's web browser, in response to HTML code provided to the customer's computer by the web server, sent an HTTP POST to the web server, which included in the POST's path information the phrase "confirm-order". *See, e.g.*, AllFilesJune1995 at src/catalog/libcatalog-access/query.c: process_form(). When the web server sent the HTTP POST to the obidos system, the `main()` function obtained the session identifier from the POST by calling `assign_uid()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main();src/website/libobidos/uid.c: assign_uid() (pulling the uid from the PATH_INFO). Then it handled the POST by calling `handle_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main(). `handle_form()` called `process_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: handle_form().
[1.3] the client initiating the conversation with the server using the stateless protocol;	The Amazon Prior Art System discloses the client initiating the conversation with the server using the stateless protocol. *See*, *e.g.*,:

Claims of the '601 Patent	The Amazon Prior Art System
	In the Amazon Prior Art System, the customer used a web browser running locally on the customer's computer to interact with services running on Amazon's computers. The customer's web browser would communicate over the Internet (using HTTP) with Amazon's web server. Amazon's web server would receive HTTP requests from the customer's computer. The web server would communicate with the Amazon obidos system to process requests from the customer and to generate HTML pages. As part of processing requests and generating web pages, obidos would communicate with databases that stored information. Obidos would send HTML pages to the web server. The web server would communicate the HTML pages to the customer's computer over the Internet (using HTTP).
[1.4] detecting when the request for a service requires preservation of the state information;	The Amazon Prior Art System discloses detecting when the request for a service requires preservation of the state information. *See*, *e.g.*,: When a customer was interacting with the Amazon Prior Art System, the obidos system assigned the customer a session identifier. Obidos included the session identifier in the HTML page sent to the user in one of two ways: (1) a hidden field in a form; or (2) as part of the path in a HREF tag. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/uid.c: assign_uid(). The session identifier was a key to an item in the sessions table in a database, which stored information about the state of the user's session. *See, e.g.*, AllFilesJune1995 at src/oracle-access/libacb/session.c: session_save(). The session identifier allowed the obidos system to maintain a user's state between HTTP responses. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main(). The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*, AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[3] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from

[3] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System

	the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp:
	
	cat_subst(stdout, webfile("user-review.html"), uid,
	isbn, title, authbuff, 0);
	
	AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id.

Claims of the '601 Patent	The Amazon Prior Art System
	Based partially on the HTML generated during the call to `cat_subst()`, the Amazon Prior Art System generated a web page that was sent to the customer. Included in the generated web page was the customer's Session ID. As an example, when the customer confirmed the order, the customer's web browser, in response to HTML code provided to the customer's computer by the web server, sent an HTTP POST to the web server, which included in the POST's path information the phrase "confirm-order". *See, e.g.*, AllFilesJune1995 at src/catalog/libcatalog-access/query.c: process_form(). When the web server sent the HTTP POST to the obidos system, the `main()` function obtained the session identifier from the POST by calling `assign_uid()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main();src/website/libobidos/uid.c: assign_uid() (pulling the uid from the PATH_INFO). Then it handled the POST by calling `handle_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main(). `handle_form()` called `process_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: handle_form().
[1.5] performing said service and identifying all continuations in an output from said service, in response to said step of detecting;	The Amazon Prior Art System discloses performing said service and identifying all continuations in an output from said service, in response to said step of detecting. *See, e.g.*,: When a customer was interacting with the Amazon Prior Art System, the obidos system assigned the customer a session identifier. Obidos included the session identifier in the HTML page sent to the user in one of two ways: (1) a hidden field in a form; or (2) as part of the path in a HREF tag. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/uid.c: assign_uid(). The session identifier was a key to an item in the sessions table in a database, which stored information about the state of the user's session. *See, e.g.*, AllFilesJune1995 at src/oracle-access/libacb/session.c: session_save(). The session identifier allowed the obidos system to maintain a user's state between HTTP responses. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main(). The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*,

Claims of the '601 Patent	The Amazon Prior Art System

	AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[4] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp:
	
	cat_subst(stdout, webfile("user-review.html"), uid,

[4] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System
	isbn, title, authbuff, 0); AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id. Based partially on the HTML generated during the call to `cat_subst()`, the Amazon Prior Art System generated a web page that was sent to the customer. Included in the generated web page was the customer's Session ID. As an example, when the customer confirmed the order, the customer's web browser, in response to HTML code provided to the customer's computer by the web server, sent an HTTP POST to the web server, which included in the POST's path information the phrase "confirm-order". *See, e.g.*, AllFilesJune1995 at src/catalog/libcatalog-access/query.c: process_form(). When the web server sent the HTTP POST to the obidos system, the `main()` function obtained the session identifier from the POST by calling `assign_uid()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main();src/website/libobidos/uid.c: assign_uid() (pulling the uid from the PATH_INFO). Then it handled the POST by calling `handle_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main(). `handle_form()` called `process_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: handle_form().
[1.6] recursively embedding the state information in all identified continuations; and	The Amazon Prior Art System discloses recursively embedding the state information in all identified continuations. *See*, *e.g.*,: The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*,

Claims of the '601 Patent	The Amazon Prior Art System

	AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[5] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp:
	
	cat_subst(stdout, webfile("user-review.html"), uid,

[5] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System
	isbn, title, authbuff, 0); AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id. Based partially on the HTML generated during the call to `cat_subst()`, the Amazon Prior Art System generated a web page that was sent to the customer. Included in the generated web page was the customer's Session ID in every link that required use of the session ID.
[1.7] communicating the output to the client, in response to said step of embedding, wherein the state information is preserved and provided to all services for the duration of the conversation.	The Amazon Prior Art System discloses communicating the output to the client, in response to said step of embedding, wherein the state information is preserved and provided to all services for the duration of the conversation. *See*, *e.g.*,: The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*, AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[6] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are

[6] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System

	excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp:
	
	cat_subst(stdout, webfile("user-review.html"), uid,
	isbn, title, authbuff, 0);
	
	AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id.
	
	Based partially on the HTML generated during the call to `cat_subst()`, the Amazon Prior Art System generated a web page that was sent to the customer. Included in the generated web page was the customer's Session ID in every link that required use of the session ID.

Claims of the '601 Patent	The Amazon Prior Art System
Claim 2	

[2.1] The method of claim 1, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding.	The Amazon Prior Art System discloses the method of claim 1, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding. *See*, *e.g.*,:
	
	The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*, AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[7] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`

[7] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp: cat_subst(stdout, webfile("user-review.html"), uid, isbn, title, authbuff, 0); AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id. Based partially on the HTML generated during the call to `cat_subst()`, the Amazon Prior Art System generated a web page that was sent to the customer. Included in the generated web page was the customer's Session ID in every link that required use of the session ID.
Claim 3	
[3.1] The method of claim 2, further comprising the step of storing at least part of the state information in a memory coupled to the server and	The Amazon Prior Art System discloses the method of claim 2, further comprising the step of storing at least part of the state information in a memory coupled to the server. *See*, *e.g.*,: When a customer was interacting with the Amazon Prior Art System, the obidos system assigned the customer a session identifier. Obidos included the session identifier in the HTML page sent to the user in one of two ways: (1) a hidden field in a form; or (2) as part of the path in a HREF tag. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/uid.c: assign_uid(). The session identifier was a key to an item in the sessions table, which stored information about the state of the user's session. *See, e.g.*, AllFilesJune1995 at src/oracle-access/libacb/session.c: session_save(). The session identifier allowed the obidos system to maintain a user's state between HTTP responses. In later versions, obidos had the ability to store the session identifier in a cookie. *See, e.g.*, AllFilesJune1996 at

Claims of the '601 Patent	The Amazon Prior Art System
	src/website/libobidos/uid.c: find_session_id() and src/website/libhttp/http-util.c: cookie_session_id().
[3.2] wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations.	The Amazon Prior Art System discloses wherein said step of embedding includes embedding an index representing said part of the state information in said all identified continuations. *See*, *e.g.*,: When a customer was interacting with the Amazon Prior Art System, the obidos system assigned the customer a session identifier. Obidos included the session identifier in the HTML page sent to the user in one of two ways: (1) a hidden field in a form; or (2) as part of the path in a HREF tag. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/uid.c: assign_uid(). The session identifier was a key to an item in the sessions table, which stored information about the state of the user's session. *See, e.g.*, AllFilesJune1995 at src/oracle-access/libacb/session.c: session_save(). The session identifier allowed the obidos system to maintain a user's state between HTTP responses. In later versions, obidos had the ability to store the session identifier in a cookie. *See, e.g.*, AllFilesJune1996 at src/website/libobidos/uid.c: find_session_id() and src/website/libhttp/http-util.c: cookie_session_id(). In addition, to the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Spinning the Web to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 4	
[4.1] The method of claim 1, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client.	The Amazon Prior Art System discloses the method of claim 1, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client. The Amazon Prior Art System discloses storage of state information in cookies. *See*, *e.g.*,: When a customer was interacting with the Amazon Prior Art System, the obidos system assigned the customer a session identifier. Obidos included the session identifier in the HTML page sent to the user in one of two ways: (1) a hidden field in a form; or (2) as part of the path in a HREF tag. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/uid.c: assign_uid(). The session identifier was a key to an item in the sessions table, which stored information about the state of the user's session. *See,*

Claims of the '601 Patent	The Amazon Prior Art System
	e.g., AllFilesJune1995 at src/oracle-access/libacb/session.c: session_save(). The session identifier allowed the obidos system to maintain a user's state between HTTP responses. In later versions, obidos had the ability to store the session identifier in a cookie. *See, e.g.*, AllFilesJune1996 at src/website/libobidos/uid.c: find_session_id() and src/website/libhttp/http-util.c: cookie_session_id(). In addition, to the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Spinning the Web to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 5	
[5.1] The method of claim 4, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information.	The Amazon Prior Art System discloses the method of claim 4, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information. The Amazon Prior Art System discloses storage of state information in cookies. *See*, *e.g.*,: When a customer was interacting with the Amazon Prior Art System, the obidos system assigned the customer a session identifier. Obidos included the session identifier in the HTML page sent to the user in one of two ways: (1) a hidden field in a form; or (2) as part of the path in a HREF tag. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/uid.c: assign_uid(). The session identifier was a key to an item in the sessions table, which stored information about the state of the user's session. *See, e.g.*, AllFilesJune1995 at src/oracle-access/libacb/session.c: session_save(). The session identifier allowed the obidos system to maintain a user's state between HTTP responses. In later versions, obidos had the ability to store the session identifier in a cookie. *See, e.g.*, AllFilesJune1996 at src/website/libobidos/uid.c: find_session_id() and src/website/libhttp/http-util.c: cookie_session_id(). In addition, to the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Spinning the Web to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.

Claims of the '601 Patent	The Amazon Prior Art System
Claim 6	
[6.1] The method of claim 1, further comprising the steps of: the client selecting a second continuation from said all identified continuations with embedded state information; and	The Amazon Prior Art System discloses the method of claim 1, further comprising the steps of: the client selecting a second continuation from said all identified continuations with embedded state information. *See*, *e.g.*,: The Amazon Prior Art System put the session ID in to links on a page as described above. When a client selected one of these links, it would cause an HTTP request including the session information. As an example, when the customer confirmed the order, the customer's web browser, in response to HTML code provided to the customer's computer by the web server, sent an HTTP POST to the web server, which included in the POST's path information the phrase "confirm-order". *See, e.g.*, AllFilesJune1995 at src/catalog/libcatalog-access/query.c: process_form(). When the web server sent the HTTP POST to the obidos system, the `main()` function obtained the session identifier from the POST by calling `assign_uid()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main();src/website/libobidos/uid.c: assign_uid() (pulling the uid from the PATH_INFO). Then it handled the POST by calling `handle_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main(). `handle_form()` called `process_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: handle_form().
[6.2] restoring the state information from said second continuation and invoking an associated second service with restored state information;	The Amazon Prior Art System discloses restoring the state information from said second continuation and invoking an associated second service with restored state information. *See*, *e.g.*,: As an example, when the customer confirmed the order, the customer's web browser, in response to HTML code provided to the customer's computer by the web server, sent an HTTP POST to the web server, which included in the POST's path information the phrase "confirm-order". *See, e.g.*, AllFilesJune1995 at src/catalog/libcatalog-access/query.c: process_form(). When the web server sent the HTTP POST to the obidos system, the `main()` function obtained the session identifier from the POST by calling `assign_uid()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main();src/website/libobidos/uid.c: assign_uid() (pulling the uid from the PATH_INFO). Then it handled the POST by calling `handle_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main(). `handle_form()` called `process_form()`. *See, e.g.*,

Claims of the '601 Patent	The Amazon Prior Art System
	AllFilesJune1995 at src/website/obidos/query-main.c: handle_form(). When a customer was interacting with the Amazon Prior Art System, the obidos system assigned the customer a session identifier. Obidos included the session identifier in the HTML page sent to the user in one of two ways: (1) a hidden field in a form; or (2) as part of the path in a HREF tag. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/uid.c: assign_uid(). The session identifier was a key to an item in the sessions table, which stored information about the state of the user's session. *See, e.g.*, AllFilesJune1995 at src/oracle-access/libacb/session.c: session_save(). The session identifier allowed the obidos system to maintain a user's state between HTTP responses. In later versions, obidos had the ability to store the session identifier in a cookie. *See, e.g.*, AllFilesJune1996 at src/website/libobidos/uid.c: find_session_id() and src/website/libhttp/http-util.c: cookie_session_id().
[6.3] recursively identifying and embedding the state information in all continuations associated with an output from said second service.	The Amazon Prior Art System discloses recursively identifying and embedding the state information in all continuations associated with an output from said second service. *See*, *e.g.*,: When a customer was interacting with the Amazon Prior Art System, the obidos system assigned the customer a session identifier. Obidos included the session identifier in the HTML page sent to the user in one of two ways: (1) a hidden field in a form; or (2) as part of the path in a HREF tag. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/uid.c: assign_uid(). The session identifier was a key to an item in the sessions table, which stored information about the state of the user's session. *See, e.g.*, AllFilesJune1995 at src/oracle-access/libacb/session.c: session_save(). The session identifier allowed the obidos system to maintain a user's state between HTTP responses. In later versions, obidos had the ability to store the session identifier in a cookie. *See, e.g.*, AllFilesJune1996 at src/website/libobidos/uid.c: find_session_id() and src/website/libhttp/http-util.c: cookie_session_id(). As another example, after the customer has entered information into the webpage generated by the Amazon Prior Art System, including the customer's e-mail address and password, the customer's web browser would send the contents of the form to the Amazon Prior Art System as a HTTP POST. This HTTP POST also included the customer's Session ID and its URL included the phrase "order-form-page1". The Amazon Prior Art System parsed the HTTP POST by the `main()` function calling `handle_form()`. *See, e.g.*, AllFilesJune1995at src/website/obidos/query-main.c: main();

Claims of the '601 Patent	The Amazon Prior Art System
	and AllFilesJune1995 at src/website/obidos/query-main.c: handle_form(). `handle_form()` called `process_form()`. *See, e.g.*, AllFilesJune1995 src/website/obidos/query.c: process_form(). `process_form()` parsed the HTTP POST and called `process_order_form_page1()` when the POST's URL contained the phrase "order-form-page1." *See, e.g.*, AllFilesJune1995 at src/catalog/libcatalog-access/query.c: process_form(). `process_order_form_page1()` attempted to look up the customer data based on the customer's e-mail address and password included in the HTTP POST by calling `access_account()`. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/order.c: process_order_form_page1() and access_account(). If the customer's account could not be accessed, then the variable `new_email` is assigned to NULL. *Id.* at access_account(). `process_order_form_page1()` then called `print_order_form()`, passing it `new_email`. *Id.* at process_order_form_page1(). `print_order_form()` called `print_order_form_header()`, passing it `new_email`. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/output.c: print_order_form(). `print_order_form_header()` used the Session ID passed into the HTTP POST to obtain a session from the Amazon database. *Id.* at print_order_form_header(). Using the session, `print_order_form_header()` retrieved the associated address object. *Id.* *See also*, AllFilesJune1995 at src/oracle-access/libacb/customer_order.h: CO_ADDRESS(). Using the address, `print_order_form_header()` obtained customer data, including the customer's name and shipping address. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/output.c: print_order_form_header(). `print_order_form_header()` called `cat_subst()`, passing it new-us-order-form-header.html (for new customers shipping to the US), old-us-order-form-header.html (for returning customers shipping to the US), new-other-order-form-head.html (for new customers shipping outside the US), or old-other-order-form-head.html (for returning customers shipping outside the US). *Id.* The customer data was also passed to `cat_subst()`. *Id.* Based partially on the HTML generated during the call to `cat_subst()`, the Amazon Prior Art System generated a web page that was sent to the customer. Included in the generated web page was the customer's Session ID. After the customer has entered information into the webpage generated by the Amazon Prior Art

Claims of the '601 Patent	The Amazon Prior Art System
	System, the customer's web browser would send the contents of the form to the Amazon Prior Art System as a HTTP POST. This HTTP POST also included the Session ID and its URL included the phrase "order-form-page2". The Amazon Prior Art System parsed the HTTP POST by the `main()` function calling `handle_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: main();and AllFilesJune1995 at handle_form(). `handle_form()` called `process_form()`. *See, e.g.*, AllFilesJune1995 at src/website/obidos/query-main.c: handle_form(). `process_form()` parsed the HTTP POST and called `process_order_form_page2()` when the POST's URL contained the phrase "order-form-page2." *See, e.g.*, src/catalog/libcatalog-access/query.c: parse_form().
Claim 7	
[7.1] The method of claim 1, further comprising the step of correlating the state information to a specific conversation.	The Amazon Prior Art System discloses the method of claim 1, further comprising the step of correlating the state information to a specific conversation. *See*, *e.g.*,: As an example, the Session ID embedded in output from The Amazon Prior Art System was used to retrieve stored session information about the conversation. For instance, `print_order_form_header()` used the Session ID passed into the HTTP POST to obtain a session from the Amazon database. *Id.* at print_order_form_header(). Using the session, `print_order_form_header()` retrieved the associated address object. *Id.* *See also*, AllFilesJune1995 at src/oracle-access/libacb/customer_order.h: CO_ADDRESS(). Using the address, `print_order_form_header()` obtained customer data, including the customer's name and shipping address. *See, e.g.*, AllFilesJune1995 at src/website/libobidos/output.c: print_order_form_header(). `print_order_form_header()` called `cat_subst()`, passing it new-us-order-form-header.html (for new customers shipping to the US), old-us-order-form-header.html (for returning customers shipping to the US), new-other-order-form-head.html (for new customers shipping outside the US), or old-other-order-form-head.html (for returning customers shipping outside the US). *Id.* The customer data was also passed to `cat_subst()`. *Id.* Based partially on the HTML generated during the call to `cat_subst()`, the Amazon Prior Art System generated a web page that was sent to the customer. Included in the generated web page was the

Claims of the '601 Patent	The Amazon Prior Art System
	customer's Session ID.
Claim 8	

[8.1] The method of claim 1, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	The Amazon Prior Art System discloses wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs. *See, e.g.*,:
	
	As described above with respect to claim 1, The Amazon Prior Art System was a web site networked via HTTP which used both HTTP POST information and hyperlinks to store state information sent to a server program.
	For example, the obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*, AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[8] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`

[8] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System

	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp:
	
	cat_subst(stdout, webfile("user-review.html"), uid,
	isbn, title, authbuff, 0);
	
	AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id.

Claim 9	
[9.1] The method of claim 8, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	The Amazon Prior Art System discloses the method of claim 8, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria. *See*, *e.g.*,: As described above with respect to claim 1, the Amazon Prior Art System filtered its data output through a program called cat_subst, which replaced variables in HTML templates with state

Claims of the '601 Patent	The Amazon Prior Art System

	information based on predetermined criteria such as the variable placeholder. The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*, AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[9] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in

[9] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System
	obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp: cat_subst(stdout, webfile("user-review.html"), uid, isbn, title, authbuff, 0); AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id.
Claim 10	
[10.1] The method of claim 8, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	The Amazon Prior Art System discloses the method of claim 8, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria. *See, e.g.*,: The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*, AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[10] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are

[10] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System

	excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp:
	
	cat_subst(stdout, webfile("user-review.html"), uid,
	isbn, title, authbuff, 0);
	
	AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id.

Claim 11	
[11.1] The method of claim 8, wherein said step of embedding further	The Amazon Prior Art System discloses the method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information

Claims of the '601 Patent	The Amazon Prior Art System

comprises the step of: modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments.	passed as arguments. *See, e.g.*,:
	
	The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*, AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[11] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C

[11] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System
	preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp: cat_subst(stdout, webfile("user-review.html"), uid, isbn, title, authbuff, 0); AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id.
Claim 12	
[12.1] The method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program.	The Amazon Prior Art System discloses the method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program. *See*, *e.g.*,: The obidos system generated HTML pages dynamically. The function cat_subst() used a HTML template file and a variable number of arguments to create a HTML page. *See, e.g.*, AllFilesJune1995 at src/utilities/libcatsubst/catsubst.c: cat_subst(). The HTML template files generally contained the file suffix of ".html" or ".cpp".[12] *See, e.g.*, AllFilesJune1996 at websrc directory. The template files had a number of placeholders that were substituted for values when processed by `cat_subst()`. For example, the HTML template file for generating a HTML page to allow a user to submit a review was user-review.html. The user-review.cpp HTML template from the AllFilesJune1996 collection is exemplary of the user-review.html file circa 1995. For example, the call to `cat_subst()` referring to user-review.html in AllFilesJune1995 used the same parameters the similar call to `cat_subst()` in AllFilesJune1996. *Compare* AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link() with AllFilesJune1996 at

[12] While the file suffix ".cpp" generally relates to C++ source files, in the case of the Amazon Prior Art System the files with this suffix in the websrc directory are HTML template files.

Claims of the '601 Patent	The Amazon Prior Art System

	src/website/libobidos/review.c: display_user_review_form(). A few lines from user-review.cpp are excerpted below:
	
	`<H1>`
	`Write Your Own Review`
	`</H1>`
	`<H4>of</H4>`
	`<H3>`
	`${2}`
	`<H4>by</H4>`
	`${3}`
	`</H3>`
	
	AllFilesJune1996 at websrc/user-review.cpp. In this excerpt, there are two placeholders: `${2}` and `${3}`. When `cat_subst()` was called, the values of `${2}` and `${3}` were passed in as variables. The user-review.cpp file would be used to create the user-review.html file by using the C preprocessor. *See also* AllFilesJune1996 at websrc/makefile. For example, the following code in obidos calls `cat_subst()` using the user-review.html file generated from user-review.cpp:
	
	cat_subst(stdout, webfile("user-review.html"), uid,
	isbn, title, authbuff, 0);
	
	AllFilesJune1995 at src/catalog/libcatalog-access/query.c: query_from_link(). The variables starting at `uid` were the values used to substitute the placeholders, indexed at position 0. In this example, `title` was index 2 and `authbuff` was index 3. `cat_subst()` substituted `${2}` for the value in `title` and `${3}` for the value in `authbuff`. Note that `${0}` is the uid or session id and that every link in the user-review webpage has the substitution for the session id.

Claim 51	
[51.1] A computerized method for preserving state information in a	The Amazon Prior Art System discloses a computerized method for preserving state information in a conversation via a stateless protocol between a client adapted to request services from one or more servers. *See* claim 1.1.

Claims of the '601 Patent	The Amazon Prior Art System
conversation via a stateless protocol between a client adapted to request services from one or more servers, the method comprising the steps of:	
[51.2] receiving a service request including state information, via the stateless protocol;	The Amazon Prior Art System discloses receiving a service request including state information, via the stateless protocol. *See* claim 1.3.
[51.3] identifying all continuations in an output from said service and recursively embedding the state information in all identified continuations, in response to said request; and	The Amazon Prior Art System discloses identifying all continuations in an output from said service and recursively embedding the state information in all identified continuations, in response to said request. *See* claims 1.5-1.6.
[51.4] communicating a response including the continuations and embedded state information, wherein the continuations enable another service request and one of the continuations must be invoked to continue the conversation.	The Amazon Prior Art System discloses communicating a response including the continuations and embedded state information, wherein the continuations enable another service request and one of the continuations must be invoked to continue the conversation. *See* claim 1.7.
Claim 52	
[52.1] The method of claim 51, wherein said embedding is performed by a server and said step of communicating is in response to said	The Amazon Prior Art System discloses the method of claim 51, wherein said embedding is performed by a server and said step of communicating is in response to said embedding step. *See* claim 2.1.

Claims of the '601 Patent	The Amazon Prior Art System
embedding step.	
Claim 53	
[53.1] The method of claim 52, further comprising the step of storing at least part of the state information in a memory coupled to the server and	The Amazon Prior Art System discloses the method of claim 52, further comprising the step of storing at least part of the state information in a memory coupled to the server. *See* claim 3.1.
[53.2] wherein embedding step includes embedding an index representing said part of the state information in said all identified continuations.	The Amazon Prior Art System discloses wherein embedding step includes embedding an index representing said part of the state information in said all identified continuations. *See* claim 3.2.
Claim 54	
[54.1] The method of claim 51, further comprising the step of dynamically downloading computer program code to the client to perform said embedding step, in response to said step of communicating the output to the client.	The Amazon Prior Art System discloses the method of claim 51, further comprising the step of dynamically downloading computer program code to the client to perform said embedding step, in response to said step of communicating the output to the client. *See* claim 4.1.
Claim 55	
[55.1] The method of claim 54, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said embedding step includes	The Amazon Prior Art System discloses the method of claim 54, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said embedding step includes embedding an index representing said part of the state information. *See* claim 5.1.

Claims of the '601 Patent	The Amazon Prior Art System
embedding an index representing said part of the state information.	
Claim 56	
[56.1] The method of claim 51, further comprising the steps of: receiving a second request associated with a second continuation from said all identified continuations with embedded state information; and restoring the state information from said second continuation and invoking an associated second service with restored state information; recursively identifying and embedding the state information in all continuations associated with an output from said second service.	The Amazon Prior Art System discloses the method of claim 51, further comprising the steps of: receiving a second request associated with a second continuation from said all identified continuations with embedded state information; and restoring the state information from said second continuation and invoking an associated second service with restored state information; recursively identifying and embedding the state information in all continuations associated with an output from said second service. *See* claims 6.1-6.3.
Claim 57	
[57.1] The method of claim 51, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol,	The Amazon Prior Art System discloses The method of claim 51, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs. *See* claim 8.1.

Claims of the '601 Patent	The Amazon Prior Art System
and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	
Claim 58	
[58.1] The method of claim 57, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	The Amazon Prior Art System discloses the method of claim 57, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria. *See* claim 9.1.
Claim 59	
[59.1] The method of claim 57, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	The Amazon Prior Art System discloses the method of claim 57, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria. *See* claim 10.1.

EXHIBIT 5

Invalidity of U.S. patent No. 5,961,601 under 35 U.S.C. § 103 by U.S. Patent No. 6,016,484 ("Williams")[1]

U.S. Patent No. 6,016,484, filed April 26, 1996 and issued on January 18, 2000, qualifies as prior art to U.S. Patent No. 5,961,601 ("'601 patent") under 35 U.S.C. § 102(a), (b), and/or (e) and renders obvious one or more claims of the '601 patent.

As described in the following claim chart, one or more claims of the '601 patent are invalid as obvious in view of Williams alone or in combination with other prior art references, including but not limited to the prior art identified in Defendant's Final Invalidity Contentions and the prior art described in the claim charts attached thereto, at least under Plaintiff's apparent infringement theory. Specifically, Defendant identifies relevant portions of the prior art in view of both (1) the Court's claim constructions and the plain meaning of the terms not construed, as well as (2) Plaintiff's infringement theory as set forth in Plaintiff's Final Infringement Contentions, even where inconsistent with the Court's claim constructions. Defendant does not agree that IBM has properly interpreted the Court's claim construction order nor that IBM has accurately identified functionality to meet each claim element. Nonetheless, to the extent understood, Defendant applies IBM's construction regarding such elements as indicated below

Claim Constructions

The Court has construed "continuation(s)" to mean "a new request which a client may send to a server, such as, for example, a hyperlink." Defendant has applied this understanding in its analysis in the chart below.

The Court has construed "all continuations in an output from said service" to mean "all new requests which a client may send to a server, such as, for example, a hyperlink, in an output from said service." Defendant has applied this understanding in its analysis in the chart below.

[1] The use of this reference or combinations of references as invalidating prior art under 35 U.S.C. §§ 102 and/or 103 may be based on Plaintiff's allegations of infringement. Defendant does not necessarily agree with the interpretations set forth in Plaintiff's infringement contentions and thus these invalidity contentions are not an admission that the accused products meet any particular claim element or infringe these claims. Moreover, nothing in these contentions should be interpreted as an acquiescence to or assertion of a particular claim construction by Defendant. In addition, nothing in these contentions should be interpreted as a position about whether any portion of the asserted claims is limiting or not. Further, by submitting these invalidity contentions, Defendant does not waive and hereby expressly reserves its right to raise other invalidity defenses, including but not limited to defenses under 35 U.S.C. § 112.

The parties agreed the proper construction of "recursively embedding the state information in all identified continuations" is "applying a process one or more times to each identified continuation to modify all identified continuations to include state information." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "conversation(s)" is "a sequence of communications between a client and server in which the server responds to each request with a set of continuations and the client always picks the next request from the set of continuations." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "filtering one of said hyperlinks and data output from said services according to a predetermined criteria" is "removing one of said hyperlinks and data output from said services according to criteria determined prior to removing." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "adding one of said hyperlinks and data to said output from said services according to a predetermined criteria" is "inserting one of said hyperlinks and data to said output from said services according to criteria determined prior to inserting." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "HTML" is "HyperText Markup Language." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "CGI program" is "Common Gateway Interface program." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "stateless protocol" is "a protocol where every request from a client to a server is treated independently of previous connections." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "client" is "a computer which issues commands to the server which performs the task associated with the command." Defendant has applied this understanding in its analysis in the chart below.

The parties agreed the proper construction of "state information" is "information about a conversation between a client and a server." Defendant has applied this understanding in its analysis in the chart below.

Claims of the '601 patent	Williams
Claim 1	

Claims of the '601 patent	Williams
[1.1] A computerized method for preserving state information in a conversation between a client adapted to request services from one or more servers which are networked via a stateless protocol to the client,	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[1.2] said services including one or more of data and programs which the client may request, wherein the conversation is a sequence of communications between the client and one or more servers for said services wherein each response from the server includes one or more continuations which enable another request for said services and wherein the client must invoke one of the continuations to continue the conversation, the method comprising the steps of:	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[1.3] the client initiating the conversation with the server using the stateless protocol;	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[1.4] detecting when the request for a service requires preservation of the state information;	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.

Claims of the '601 patent	Williams
[1.5] performing said service and identifying all continuations in an output from said service, in response to said step of detecting;	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[1.6] recursively embedding the state information in all identified continuations; and	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[1.7] communicating the output to the client, in response to said step of embedding, wherein the state information is preserved and provided to all services for the duration of the conversation.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 2	
[2.1] The method of claim 1, wherein said step of embedding is performed by the server and said step of communicating is in response to said step of embedding.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 3	
[3.1] The method of claim 2, further comprising the step of storing at least part of the state information in a memory coupled to the server and	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[3.2] wherein said step of embedding includes embedding an index representing said part of the	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as

Claims of the '601 patent	Williams
state information in said all identified continuations.	identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 4	
[4.1] The method of claim 1, further comprising the step of dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client.	Williams discloses dynamically downloading computer program code to the client to perform said step of embedding which is responsive to said step of communicating the output to the client. *See, e.g.*,: Unlike HTML, Java supports the notion of client-side validation, offloading appropriate processing onto the client for improved performance. Dynamic, real-time Web pages can be created. Using the above-mentioned custom UI components, dynamic Web pages can also be created. 10:3-5. Sun's Java language has emerged as an industry-recognized language for "programming the Internet." Sun defines Java as: "a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, dynamic, buzzword-compliant, general-purpose programming language. Java supports programming for the Internet in the form of platform-independent Java applets." Java applets are small, specialized applications that comply with Sun's Java Application Programming Interface (API) allowing developers to add "interactive content" to Web documents (e.g. simple animations, page adornments, basic games, etc.). Applets execute within a Java-compatible browser (e.g. Netscape Navigator) by copying code from the server to client. From a language standpoint, Java's core feature set is based on C++. Sun's Java literature states that Java is basically "C++, with extensions from Objective C for more dynamic method resolution". 10:13-25 FIG. 2 provides an alternative preferred embodiment in accordance with a Java based implementation of the invention. Java is a network application enabler for applications that utilize the Internet and HTML. Java is an object-oriented langugage that was developed by Sun Microsystems to speed development of their network applications. Only the differences between the native embodiment discussed earlier and the Java version is addressed to clarify the discussion. Accordingly, the browser 140, wallet manager 150, certificate manager 152, data manager 156, crypto & protocol library 157 and wallets 158 are

Claims of the '601 patent	Williams
	the same as described in reference to FIG. 1B. A Payment Instruction Applet 200 at the Merchant Web Site 180 is responsible for delivering the order information to the PayWindow Applet 230 on the consumer's desktop 186. This order information has the same information that is contained in the Payment Instruction MIME message which was described in the PayWindow System View Native Code section. The Pay Instruction Applet 200 is a part of the payment HTML 192 page which is displayed by the merchant to the consumer. The Pay Instruction applet 200 requires that the following parameters be set with appropriate data when the payment page is generated by the merchant system GSO, Shipping Address, Merchant, Merchant Certificate, Merchant URL and Button Text. The Pay Instruction Applet 200 displays a button called "Pay", "Use PayWindow" or the value of the Button Text parameter. Once clicked, Pay Instruction Applet 200 delivers the above information to the Pay Window Applet 230 on the customer's desktop 186. The consumer interacts with the Pay window Applet 230 Applet to complete the payment transaction. The Address Requisition Applet 204 at the Merchant Web Site 180 is utilized by the merchant system 180 to obtain a consumer's shipping and/or billing address. This applet is displayed on the HTML page which accepts the consumer's shipping or billing address. The Address Requisition Applet 204 displays a button called "V-wallet" or the value of the Button Text parameter. Once clicked, the address Requisition Applet 204 launches the AddressWindow applet 240 utilizing the consumer's address. The AddressWindow applet 240 interacts with the consumer and send the address information to the merchant system 180. The address information is then processed consistent with the processing in the native system. The Certificate Issuance CGI scripts 162 and the Single Account Wallet 160 at the Bank Web Site 182 is processed as described in the native system. The Certificate Installation Applet 220 of the Bank Web Site 182 is utilized by the Certificate Issuance CGI scripts 162 system to deliver a consumer's certificate to the consumer's desktop. A variety of applets are provided at the Veri Fone web site 184 to make the consumer's shopping experience easy and efficient. These helper applications include a Setup Applet 212, PayWindow Applet 210 and an AddressWindow Applet 214 similar to the PayWindow Helper Application Native section discussed above. 12:55-13:42.

Claims of the '601 patent	Williams
	See Figs. 31 and 32.
Claim 5	
[5.1] The method of claim 4, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said step of embedding includes embedding an index representing said part of the state information.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 6	
[6.1] The method of claim 1, further comprising the steps of: the client selecting a second continuation from said all identified continuations with embedded state information; and	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[6.2] restoring the state information from said second continuation and invoking an associated second service with restored state information;	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[6.3] recursively identifying and embedding the state information in all continuations associated with an output from said second service.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 7	
[7.1] The method of claim 1,	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person

Claims of the '601 patent	Williams
further comprising the step of correlating the state information to a specific conversation.	of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 8	
[8.1] The method of claim 1, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 9	
[9.1] The method of claim 8, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 10	
[10.1] The method of claim 8, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 11	
[11.1] The method of claim 8, wherein said step of embedding	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include

Claims of the '601 patent	Williams
further comprises the step of: modifying an identified continuation which is a request for an HTML file to invoke a CGI converter program with the identified continuation and the state information passed as arguments.	this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 12	
[12.1] The method of claim 8, wherein said step of embedding further comprises the step of: modifying an identified continuation which is an invocation to a CGI program with the identified continuation and the state information passed as arguments, wherein said step of embedding is performed by the CGI program.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 51	
[51.1] A computerized method for preserving state information in a conversation via a stateless protocol between a client adapted to request services from one or more servers, the method comprising the steps of:	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[51.2] receiving a service request including state information, via the stateless protocol;	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[51.3] identifying all continuations	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person

Claims of the '601 patent	Williams
in an output from said service and recursively embedding the state information in all identified continuations, in response to said request; and	of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[51.4] communicating a response including the continuations and embedded state information, wherein the continuations enable another service request and one of the continuations must be invoked to continue the conversation.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 52	
[52.1] The method of claim 51, wherein said embedding is performed by a server and said step of communicating is in response to said embedding step.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 53	
[53.1] The method of claim 52, further comprising the step of storing at least part of the state information in a memory coupled to the server and	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
[53.2] wherein embedding step includes embedding an index representing said part of the state information in said all identified continuations.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 54	
[54.1] The method of claim 51,	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person

Claims of the '601 patent	Williams
further comprising the step of dynamically downloading computer program code to the client to perform said embedding step, in response to said step of communicating the output to the client.	of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 55	
[55.1] The method of claim 54, further comprising the step of storing at least part of the state information in a memory coupled to the client and wherein said embedding step includes embedding an index representing said part of the state information.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 56	
[56.1] The method of claim 51, further comprising the steps of: receiving a second request associated with a second continuation from said all identified continuations with embedded state information; and restoring the state information from said second continuation and invoking an associated second service with restored state information; recursively identifying and embedding the state information in	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.

Claims of the '601 patent	Williams
all continuations associated with an output from said second service.	
Claim 57	
[57.1] The method of claim 51, wherein the client and the server are networked via the World Wide Web, the stateless protocol is hypertext transfer protocol, and the continuations are hyperlinks to one of hypertext markup language files and common gateway interface programs.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 58	
[58.1] The method of claim 57, further comprising the step of filtering one of said hyperlinks and data output from said services according to a predetermined criteria.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.
Claim 59	
[59.1] The method of claim 57, further comprising the step of adding one of said hyperlinks and data to said output from said services according to a predetermined criteria.	To the extent this limitation is not explicitly disclosed, it would have been obvious to a person of ordinary skill in the art at the time the patent-in-suit was filed to modify Williams to include this limitation in light of numerous other prior art patents, publications, articles, and systems as identified in the Combination Chart for the '601 Patent, Exhibit C-00. *See also* Final Invalidity Contentions Section I.A.3.